VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA

EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Databázová aplikace pro podporu šachové hry

Database Application for Support of the Chess Game

Student: Bc. Pavel Rakús

Vedoucí diplomové práce: Ing. Vítězslav Novák, Ph.D.

Ostrava 2014

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání diplomové práce

Student: **Bc. Pavel Rakús**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 1802T001 Aplikovaná informatika

Téma: Databázová aplikace pro podporu šachové hry
Database Application for Support of the Chess Game

Zásady pro vypracování:

1. Úvod
2. Teoretická východiska práce
3. Návrh a realizace SQL databáze a aplikační logiky v jazyce JAVA
4. Využití aplikace v praxi a srovnání s ostatními produkty
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

HEROUT, Pavel. *Java – grafické uživatelské prostředí a čeština.* 2. vyd. České Budějovice: Kopp, 2012. ISBN 978-80-7232-328-9.
LACKO, Luboslav. *Mistrovství v SQL Server 2012.* Brno: Computer Press, 2013. ISBN 978-80-251-3773-4.
SARANG, Poornachandra. *Java programming.* New York: McGraw-Hill, 2012. ISBN 978-0-07-163360-4.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Vítězslav Novák, Ph.D.**

Datum zadání: 22.11.2013
Datum odevzdání: 25.04.2014

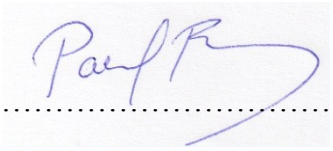
VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA OSTRAVA Ekonomická fakulta 1

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Prohlašuji, že jsem celou práci, včetně všech příloh, vypracoval samostatně.

Přílohy č. 1 a 2, dané mi k dispozici, jsem samostatně doplnil.



……

Bc. Pavel Rakús

V Ostravě dne 25. 4. 2014

Na tomto místě bych rád poděkoval vedoucímu této diplomové práce, panu Ing. Vítězslavu Novákovi, Ph.D., za odborné vedení, relevantní připomínky a také ochotu a vstřícnost při našem společném jednání.

Obsah

1 Úvod

Šachová hra prošla za celou svou historii spoustou změn. Ve všech zápasech na vrcholné úrovni byly informace o soupeři velmi ceněnou veličinou, vždyť i Bobby Fischer, mnohými označovaný za nejlepšího šachistu všech dob, by se jen těžko stal mistrem světa, kdyby nečerpal o svých soupeřích důležité informace z desítek a stovek vydání časopisů a šachových knih. Skutečná revoluce v této královské hře se ale odehrála na konci 20. století. S příchodem nových počítačových programů a rozsáhlých databází se od základů změnila filozofie šachistů při přípravě k jednotlivým partiím. Velkou zásluhu na tom mělo i slavné vítězství počítačového programu Deep Blue nad Garri Kasparovem v roce 1997. Nyní, v roce 2014, byla šachová hra dovedena do takového stavu, že z každé významné události je na internet přenášen on - line průběh partií, často i s analýzou silných šachových motorů, a následně jsou partie pro všechny zájemce volně ke stažení na webových stránkách pořadatele. Tyto data pak představují opravdu cenný zdroj informací pro budoucí soupeře hráčů, kteří partie sehráli. Vždyť kdo by nechtěl vědět, jakým tahem soupeř zahajuje partii, jak reaguje v taktických zápletkách, jaké volí varianty zahájení a další podrobnosti. Takové informace přináší každému šachistovi před začátkem partie velkou výhodu.

Cílem této práce je popsat teoreticky i metodicky postup při tvorbě přehrávače šachových partií využívajícího relační databáze v SQL, který umožní uživateli vyhledat a přehrát šachovou partii získanou z této databáze nebo nahranou ze souboru ve formátu PGN.

Práce je členěna do tří hlavních kapitol a dále do dalších podkapitol. V první kapitole jsou postupně popsány základní teoretické předpoklady, na kterých je práce založena. Ve druhé kapitole jsou uvedeny případy užití aplikace jednotlivými uživateli, podrobně popsána struktura relační databáze a samotná aplikační logika Java Enterprise Edition aplikace. Ve třetí kapitole je věnován prostor srovnání aplikace s dalším šachovým softwarem a je naznačeno využití aplikace v praxi.

2 Teoretická východiska práce

V této části práce jsou postupně popsány a shrnuty teoretické předpoklady, na kterých je tato diplomová práce založena. V úvodní části je kapitola věnována programovacímu jazyku Java, počínaje historií a vývojem, až po konkrétní popis základních prvků Java Enterprise Edition, použitých při tvorbě samotné aplikace. Následně je nastíněna základní problematika relačních SQL databází a v závěru kapitoly pak stručný popis formátu PGN, jenž je do velké míry v aplikaci využíván, a jehož specifikace je pro samotný běh aplikace zásadní.

2.1 Základní teorie a filozofie programovacího jazyka Java

Tato kapitola je zaměřena na základní teorii a filozofii programovacího jazyka *Java* – jeho historii, vývoj, srovnání s příbuznými programovacími jazyky, přínos pro internet, základní hesla a celkově obecně vzato základní filozofii tohoto velmi úspěšného a rozšířeného nástroje.

2.1.1 Počátky Javy

Programovací jazyk byl navrhnut a zkonstruován v roce 1991 Jamesem Goslingem, Patrickem Naughtonem, Chrisem Warthem, Edem Frankem a Mikem Sheridanem ze společnosti Sun Microsystems. Původní název nového programovacího jazyka byl *„Oak“*, v roce 1995 ovšem došlo k jeho přejmenování na současnou podobu Java. Hlavním cílem těchto vývojářů bylo uspokojení potřeby jazyka nezávislého na platformě, který by bylo možné využít pro tvorbu softwaru, vsazovaného do nejrůznějších zařízení spotřební elektroniky jako jsou mikrovlnné trouby, dálkové ovladače apod. K činnosti těchto zařízení je jako řadičů využíváno mnoho různých typů výpočetních jednotek. Většina počítačových jazyků v té době byla navržena tak, aby se kompilovala pro určitý cíl. Problémem ovšem byla samotná tvorba kompilátorů, která byla drahá a časově náročná. Ve snaze nalézt lepší řešení proto začal Gosling s kolegy pracovat na přenositelném jazyce, nezávislém na platformě, který by bylo možné spustit na rozličných výpočetních jednotkách a v různých prostředích. A právě toto úsilí bylo počátkem jazyka Java. Dalším velmi důležitým faktorem pro budoucí úspěch Javy se stal zrod celosvětové webové sítě (WWW). Díky potřebě přenositelnosti programů byla s rozvojem webu Java katapultována do popředí návrhu počítačových jazyků, kde se udržuje dodnes. [9]

2.1.2 Vývoj Javy

Jak již bylo uvedeno, první vydání jazyka Java s označením 1.0 proběhlo v roce 1995 společností Sun Microsystems. V průběhu let byla Java dále vyvíjena, díky oblibě byl zajištěn její růst a také redefinování. Java byla také několikrát upgradována – některé upgrady byly pouze kosmetické, jiné měly podobu mnohem významnější. [9]

Za první výraznou aktualizaci lze považovat vydání verze 1.1. A ačkoliv v této verzi bylo uvedeno relativně málo změn oproti verzi předchozí, nově uvedené vlastnosti byly pro Javu velmi významné, neboť jejich přidáním byla výrazně zvýšena funkčnost a elegantnost jazyka. Mezi tyto nové vlastnosti můžeme zařadit:

- přidání tzv. vnitřních (skrytých) tříd (*inner classes*),
- možnost deklarovat lokální proměnné a parametry metod pomocí modifikátoru `final`,
- přidání nového druhu primárních výrazů s označením *literály tříd* atd. [1]

Za další významné vydání Javy je možné považovat verzi Java 2, kde 2 je označení pro tzv. „druhou generaci“. První vydání Javy 2 bylo označeno jako verze 1.2. Důvodem pro toto označení byl původní vztah k internímu číslu verze knihoven Javy, později bylo toto zobecněno tak, aby se označení vztahovalo k vydání jako celku. S verzí Java 2 byl společností Sun Microsystems výsledný produkt nazván J2SE (Java 2 Platform Standard Edition) a čísla verzí začala být aplikována na tento produkt. Dalším upgradem byla verze J2SE 1.3, následovaná verzí J2SE 1.4, kterou byla dále vylepšena funkčnost a v níž bylo obsaženo několik důležitých nových prvků jako zřetězené výjimky, kanálově založené vstupně – výstupní operace a klíčové slovo `assert`. [9]

Za druhou revoluci v Javě je ovšem považováno vydání J2SE 5. Na rozdíl od většiny předchozích upgradů, přinášejících důležitá, avšak pouze přírůstková vylepšení, byl vydáním J2SE 5 v zásadě rozšířen záběr, síla i rozsah tohoto jazyka. Pro představu lze uvést, že mezi nové prvky byly zařazeny položky jako:

- generické typy,
- automatické zabalování a rozbalování,
- výčty,

- variabilní počet argumentů (varargs),
- vylepšený cyklus `for` ve stylu „`for-each`“,
- statické importování,
- anotace.

Tento seznam není pouze výčtem drobných úprav nebo inkrementálních upgradů. Každou položkou je představován výrazný přírůstek k jazyku Java. Ať se již jedná o generické typy, rozšířený příkaz `for` nebo variabilní počet argumentů, kterými byly navíc zavedeny nové syntaktické prvky, či automatické zabalování a rozbalování, jimiž byla upravena sémantika jazyka. *Anotacemi* byl do programování vnesen zcela nový rozměr. Význam nových prvků je také naznačen v použitém čísle verze „5“. Následující číslo pro další verzi by bylo obvyklé 1.5, nicméně nové prvky byly tak podstatné, že bylo společností Sun rozhodnuto o zvýšení čísla verze na „5“ a tím i o zdůraznění této významné události. Další vydání bylo nazváno Java SE 6, kdy bylo rozhodnuto o vypuštění čísla „2“ z názvu. Vydání Java SE 7, se sadou JDK 7 (sada pro vývojáře) a interním číslem 1.7 je první hlavní vydání Javy od doby, kdy společnost Sun Microsystems ovládla společnost Oracle (ten proces byl uskutečněn od dubna 2009 do ledna 2010). Tuto verzi v příslušném updatu lze považovat za aktuální. [9]

2.1.3 Srovnání Javy s jazyky C, C++ a C#

Jazyk Java je přímo souvislý s jazyky C a C++, protože z jazyka C byla převzata jeho syntaxe a objektový model byl převzat z jazyka C++. Vztah Javy a jazyků C a C++ je důležitý hned z několika hledisek. Programátor se znalostí jazyka C/C++ je schopen se snadno naučit jazyk Java a naopak. V Javě bylo převzato vysoce úspěšné programovací paradigma, reprezentované právě těmito jazyky. I přesto, že existuje mnoho podobností mezi C++ a Javou, zvláště pak podpora objektově orientovaného programování, nejsou tyto jazyky v žádném směru kompatibilní. Mylná je proto představa Javy jako jakési „internetové verzi jazyka C++“. [9]

Jazyk C# byl vyvinut společností Microsoft několik let po vytvoření Javy. U obou těchto jazyků je možné pozorovat úzkou souvislost. Syntaxe obou je ve stylu jazyka C++, u obou je podporováno distribuované programování a využíván stejný objektový model. I přesto není pravděpodobné úplné nahrazení Javy jazykem C#, např. také proto, že oba jsou optimalizovány pro dva odlišné typy výpočetních prostředí. [9]

2.1.4 Bajtový kód Javy

Bajtový kód je vysoce optimalizovaná sada instrukcí navržených k provádění běhovým systémem Javy, který se nazývá virtuální stroj Javy (z angl. *Java Virtual Machine* - JVM). Původně byl JVM navržen jako interpret pro bajtový kód, což může být částečně překvapující i vzhledem k tomu, že řada moderních jazyků je kvůli výkonu navržena pro kompilování do spustitelného kódu. Program v Javě přeložený do bajtového kódu lze díky běhu přes JVM snadněji spouštět v celé řadě prostředí, pro něž stačí pouze to, aby byl implementován právě virtuální stroj Javy. Na takovémto systému s tímto běhovým balíčkem lze následně spustit libovolný program v Javě. Platí, že rychlost programu při interpretování je obecně nižší než u stejného programu při zkompilování do spustitelného kódu. V případě Javy je ovšem bajtový kód vysoce optimalizován, čímž je virtuálnímu stroji Javy umožněno rychlejší provádění programu, než by se na první pohled mohlo zdát. [9]

2.1.5 Základní hesla Javy

I přesto, že za základní síly, jež vedly k vytvoření Javy, je možné považovat hlavně přenositelnost a bezpečnost, významný podíl nesmí být upřen ani dalším faktorům. Klíčová kritéria byla shrnuta týmem odpovědným za návrh Javy do následujícího seznamu základních hesel Javy, pomocí kterého je deklarováno, že jazyk Java je:

- *jednoduchý* (tzn. se stručnou a kompaktní sadou rysů, jež usnadňují jeho osvojování a používání),
- *bezpečný* (tzn., že jím jsou poskytovány bezpečné prostředky pro tvorbu internetových aplikací),
- *přenositelný* (tzn. možnost provádět a spouštět programy v Javě v libovolném prostředí, pro které existuje běhový systém Javy),
- *objektově orientovaný* (tzn. ztělesnění moderní filozofie programování),
- *robustní* (tzn., že jím je podporováno bezchybné programování, neboť je striktně typován a je jím prováděna kontrola za běhu),
- *vícevláknový* (tzn., že jím je poskytována podpora vícevláknového programování),
- *architektonicky neutrální* (tzn., že není svázán s architekturou určitého stroje či operačního systému),

- *interpretovaný* (tzn., že jím je podporován kód nezávislý na platformě),
- *vysoce výkonný* (tzn., že bajtový kód Javy je vysoce optimalizován na rychlost provádění),
- *distribuovaný* (tzn., že byl navržen s ohledem na distribuované prostředí Internetu),
- *dynamický* (tzn., že u programů je využíváno množství běhových informací o typech, jenž jsou za běhu používány pro ověřování a vyhodnocování přístupu k objektům). [9]

2.1.6 Objektově orientované programování v Javě

Objektově orientované programování (OOP) je srdcem jazyka Java. OOP je představován silný způsob přístupu k programování, když byly v jeho filozofii zkombinovány nejlepší myšlenky ze strukturovaného programování s několika novými koncepcemi, čímž bylo ve výsledku dosaženo odlišného způsobu uspořádání programu. Kvůli podpoře principů OOP je u všech objektově orientovaných jazyků možné pozorovat tři společné rysy: *zapouzdření, polymorfizmus a dědičnost*. [9]

Zapouzdření

Zapouzdření je programovací mechanismus, jehož pomocí je kód svázán s daty, se kterými je v součinnosti, a kterým je udržován kód i data v bezpečí před narušením a nesprávným použitím zvenčí. Za prostředek, jenž podporuje zapouzdření, může být považován *objekt*. Pro zjednodušení si ho lze představit jako samostatnou černou skříňku, ve které jsou kombinována veškerá nezbytná data a zdrojový kód. [9]

Polymorfizmus

Polymorfizmus (z řečtiny, význam je „mnoho forem“) je vlastnost, jejímž prostřednictvím je umožněno jedním rozhraním přistupovat k obecné třídě akcí. Lze se také setkat s obecným vyjádřením polymorfizmu obratem „jedno rozhraní, více metod“. To znamená, že je možno pro skupinu souvisejících činností navrhnout obecné rozhraní, čím je dosahováno nižší složitost programu. [9]

Dědičnost

Dědičnost je proces, při kterém je jednomu objektu umožněno získat vlastnosti jiného objektu. Tato vlastnost je velmi důležitá, její pomocí je podporován princip hierarchické

klasifikace. Bez hierarchií by musely být u každého objektu explicitně definovány všechny jeho charakteristiky. Díky dědičnosti je postačující objektu definovat jenom ty vlastnosti, kterými je ve své třídě jedinečným. [9]

2.2 Základní mechanismy programovacího jazyka Java

V této kapitole jsou shrnuty veškeré základní podklady pro úspěšnou tvorbu programu v Javě. Jedná se o rozšiřující datové typy (třída, pole a řetězec), principy fungování metod, mechanismus zpracování výjimek a také možnost použití vstupně – výstupních operací.

2.2.1 Třídy

Třídy jsou jádrem jazyka Java. Třída je považována za základ, na němž je postaven celý jazyk Java, protože její pomocí je definována povaha objektu. Samotnými třídami je pak tvořen základ pro objektově orientované programování v Javě. Uvnitř tříd jsou definována data a kód, který nad těmito daty pracuje, a který je obsažen v různých metodách. [8]

Třída je šablona, pomocí níž je definován tvar objektu. Jsou stanovena data i kód, který bude na těchto datech pracovat. V Javě je využívána specifikace třídy pro konstrukci objektů. Objekty jsou instance nějaké třídy. Třída je tedy v podstatě skupina plánů, jež stanoví, jak sestavit objekt, logická abstrakce. V paměti je fyzicky reprezentována až v okamžiku, kdy je vytvořen její objekt. [9]

Třída je vytvářena pomocí klíčového slova `class`. Zjednodušený obecný tvar definice třídy může mít následující podobu:

```
class NazevTridy {
 // deklarace proměnných instance
 typ promenna1;
 typ promenna2;
 ...
 typ promennaN;

 // deklarace metod
 navratovyTyp metoda1(parametry) {
       příkazy;
 }
 navratovyTyp metoda2(parametry) {
       příkazy;
 }
 ...
 navratovyTyp metodaN(parametry) {
       příkazy;
 }
}
```

Tento zápis je samozřejmě velkým zjednodušením. Definice třídy pomocí klíčového slova `class` téměř vždy obsahuje modifikátory přístupu – `public, private` nebo `protected`. Pokud není využit žádný z těchto modifikátorů, předpokládá se použití výchozího nastavení.

Pokud je člen třídy uveden jako `private`, je přístup k němu povolen pouze ostatním členům této třídy. Metodám v ostatních třídách není povoleno přistupovat k soukromému členu jiné třídy. Přístup k těmto členům je pak obvykle realizován pomocí speciálních metod, které jsou označovány jako *getter* a *setter*. Obecně platí pravidlo, že datové položky by měly být zapouzdřeny a přistupovat by se k nim mělo právě pomocí těchto metod. [9]

V definici třídy mohou být uvedena také další klíčová slova – `abstract`, definující abstraktní třídu, `extends`, indikující rozšíření jiné třídy či `implements`, naznačující využití rozhraní.

2.2.2 Objekty

Definicí třídy je vytvářen nový datový typ. Jak již bylo zmíněno, objekty jsou vždy instancí nějaké libovolné třídy. Deklarace třídy je pouze popisem typu a není takto vytvářen

skutečný objekt. Pro samotné vytvoření objektu je nutné využít příkazu v následující ilustrativní podobě:

$$Partie\ sachovaPartie = new\ Partie(parametry);$$

V této deklaraci jsou prováděny dvě věci. Nejdříve je deklarována proměnná s názvem `sachovaPartie` a s typem třídy `Partie`. Tato proměnná objekt nedefinuje, jedná se o proměnnou, která na objekt odkazuje. Poté je touto deklarací vytvořena fyzická kopie objektu a proměnné `sachovaPartie` přiřazena reference na tento objekt. K této činnosti je využíváno operátoru `new`. Operátor `new` dynamicky (za běhu) alokuje paměť pro objekt a vrací na něj referenci (odkaz). Tato reference může být považována za adresu objektu v paměti, alokovanou právě operátorem `new`. Reference je následně uložena do proměnné. V jazyce Java je tedy nutné všechny instance tříd dynamicky alokovat. [9]

2.2.3 Metody

Metody jsou podrutiny, pomocí kterých je manipulováno s daty definovanými třídou, a pomocí kterých je v řadě případů poskytován přístup k těmto datům. Ve velké většině případů je komunikace ostatních částí programu se třídou povolena právě pouze prostřednictvím metod. [7]

Metoda obsahuje vždy jeden či více příkazů a měla by vždy provádět pouze jednu danou úlohu. Obecně platí, že metodě může být přiřazeno libovolné jméno – samozřejmě mimo klíčových slov Javy a názvu `main()`, jenž je rezervován pro metodu zahajující provádění programu. Obecný tvar metody je definován v následující podobě:

```
navratovyTyp nazev(seznamParametru) {
 příkazy;
}
```

Výrazem `navratovyTyp` je zde stanoven typ dat, který je danou metodou navrácen. Může jít o libovolný platný typ, včetně třídních typů, které byly vytvořeny uživatelem. Pokud není metodou vracena žádná hodnota, její návratový typ musí být nastaven na `void`. Název metody je nastaven pomocí výrazu `nazev`. Může se jednat o libovolný platný identifikátor odlišný od ostatních prvků, které jsou již v aktuálním oboru použity. Výraz `seznamParametru` je posloupnost dvojic tvořených typem a identifikátorem, které jsou

odděleny čárkami. Parametry jsou proměnné, ke kterým je předána hodnota argumentů získaných metodou při jejím volání. Seznam parametrů může být i prázdný, v takovém případě metoda nemá žádné parametry.[9]

Pokud je metoda typu `void`, může být návrat způsoben při uzavírací složené závorce metody (tedy po provedení všech příkazů v těle metody) nebo v průběhu běhu programu pomocí provedení příkazu `return`. Častěji jsou ovšem využívány metody, které mají nastaven určitý návratový typ. V takovém případě je nutné za příkaz `return` doplnit ještě výraz hodnota (`return hodnota;`). [9]

Metody lze také přetěžovat. Přetížené metody (*overloaded*) jsou metody, které mají stejná jména, ale různé hlavičky. To, že je metoda přetížená, znamená, že její formální parametry musí být odlišeny svým počtem nebo typem nebo pořadím, případně i kombinacemi těchto způsobů. Metoda ovšem nemůže být přetížena pouhou změnou typu návratové hodnoty, je nutné to zařídit minimálně jedním z výše uvedených způsobů. Pokud je přetížená metoda zavolána, je kompilátorem zvolena právě ta z metod, která přesně vyhoví počtu, typům a pořadí skutečných parametrů. Přetěžování metod je využíváno většinou u funkčně příbuzných metod, které jsou odlišeny pouze parametry. Uživateli je umožněno provádět stejnou operaci s různými datovými typy, aniž by se musel zabývat doplňkovými činnostmi jako např. přetypováním parametrů apod. [3]

Konstruktory

Konstruktor je speciální typ metody, který má stejný název jako třída, a jehož úlohou je inicializovat objekt při jeho vytvoření. U konstruktoru není specifikován žádný explicitní návratový typ. Je obvykle využíván pro přiřazení počátečních hodnot do proměnných instance, definovaných danou třídou, nebo pro provedení jakýchkoliv jiných startovacích procedur, nezbytných pro vytvoření plně zformovaného objektu. Konstruktorem disponují všechny třídy, protože jazykem Java je automaticky poskytován výchozí konstruktor, který inicializuje všechny členské proměnné na jejich výchozí hodnoty, což je nula, `null`, resp. `false` pro číselné typy, referenční typy, resp. typ `boolean`. Jakmile je uživatelem definován vlastní konstruktor, výchozí konstruktor již využíván není. Konstruktor může být bezparametrický, případně může stejně jako metoda obsahovat seznam parametrů. [9]

V těle konstruktoru je velmi často využíváno klíčového slova `this`, které má speciální využití v přetížených konstruktorech. Pomocí něj lze v konstruktoru vyvolat jiný konstruktor

stejné třídy. Klíčovým slovem `this` je také nazýván odkaz na objekt, na němž byla daná metoda zavolána. V praxi se často používá pro odlišení lokální proměnné od proměnné instance a to právě v konstruktoru. [3]

2.2.4 Pole

Pole je kolekce proměnných téhož typu, na něž se lze odkázat společným jménem. V Javě mohou být pole jednorozměrná nebo vícerozměrná a jsou využívána pro nejrůznější účely, protože představují pohodlný prostředek pro seskupování souvisejících proměnných. Nejčastěji využívaným polem je pole jednorozměrné, pro jehož deklarování může být použit následující obecný tvar:

```
typ nazevPole[] = new typ[velikost];
```

Výrazem `typ` je deklarován typ prvku pole, běžně označovaný jako základní typ. Typem prvku je určen datový typ každého prvku, obsaženého v poli. Počet prvků, které mají být v dané poli uchovávány, je určen výrazem `velikost`. Pole jsou implementovány jako objekty, proto je jejich tvorba dvou-krokový proces. V prvním kroku je pole deklarováno referenční proměnnou pole. V druhém kroku je alokována paměť pro pole a odkaz na tuto paměť je přiřazen do proměnné pole. Pole jsou tedy v Javě dynamicky alokována pomocí operátoru `new`. K jednotlivým prvkům v poli se přistupuje pomocí indexu. Index popisuje pozici prvku uvnitř pole, je uveden v hranatých závorkách. V Javě je indexem prvního prvku všech polí nula. [9]

Délku pole (velikost či počet prvků) lze kdykoliv zjistit pomocí členské proměnné se jménem *length*. Tato konstanta je obsažena automaticky v jakémkoliv poli a je přístupná pouze pro čtení – z toho logicky vyplývá, že délku již inicializovaného pole nelze měnit. [3]

2.2.5 Řetězce

Na rozdíl od jiných programovacích jazyků není v Javě řetězec definován jako pole jednotlivých znaků, ale jako objekt. Třída `String` (z balíčku `java.lang`) definuje a podporuje znakové řetězce a obsahuje celou řadu metod, které jsou pro práci s řetězci využívány. V tabulce jsou uvedeny ty nejdůležitější a nejčastěji používané z nich.

Tabulka 2.1 - Nejběžnější metody objektu typu String

Metoda	Popis
`boolean equals(retezec)`	Vrací true, pokud vyvolávající řetězec obsahuje stejnou posloupnost znaků jako `retezec`.
`int length()`	Vrací délku řetězce.
`char charAt(index)`	Vrací znak na zadané indexové pozici.
`int indexOf(retezec)`	Vyhledá ve vyvolávajícím řetězci podřetězec zadaný jako `retezec`. Vrací index první nalezené shody nebo -1 v případě nenalezení.
`int lastIndexOf(retezec)`	Vyhledá ve vyvolávajícím řetězci podřetězec zadaný jako `retezec`. Vrací index poslední nalezené shody nebo -1 v případě nenalezení.
`boolean contains(retezec)`	Vrací true, pokud vyvolávající řetězec obsahuje znak(y) z parametru `retezec`.
`String subString(begin, end)`	Vrací nový řetězec od zadané startovní pozice (`begin`) až po konečnou pozici vyvolávajícího řetězce (`end`).

Pomocí dalších metod lze například řetězec převést na malá či velká písmena, ořezat z něj bílé znaky, nahradit jeho část jiným řetězcem atd. Vzhledem k tomu, že obsah objektů typu `String` je neměnitelný, je v případě nutnosti použití variace na již existující řetězec vytvořen řetězec nový s tím, že již nepoužívané objekty typu `String` jsou automaticky uvolněny z paměti. [9]

2.2.6 Výjimky

Výjimka je chyba, k níž dochází za běhu programu. Pomocí subsystému Javy pro zpracování výjimek lze strukturovaným a řízeným způsobem ošetřovat chyby vzniklé za běhu programu. Za principiální výhodu zpracování výjimek lze označit to, že je automatizována většina kódu pro ošetřování chyb, který bylo nutné do jakéhokoliv rozsáhlého programu dříve nutné zadávat „ručně“. [9]

V Javě jsou všechny výjimky reprezentovány jako třídy. Všechny třídy výjimek jsou odvozeny od třídy s názvem `Throwable`. Pokud tedy dojde v programu k nějaké výjimce, je

vygenerován objekt některého z typů tříd výjimek. Třída `Throwable` má dvě přímé podtřídy: `Exception` a `Error`. Výjimky typu `Error` mají souvislost s chybami, k nimž dochází v samotném virtuálním stroji Javy. Chyby, které jsou výsledkem činnosti programu, jsou reprezentovány jako podtřídy třídy `Exception`. Do této kategorie lze zařadit např. dělení nulou, překročení hranice pole, chyby při práci se soubory nebo chyby při převodu znaků na čísla. Důležitou podtřídou třídy `Exception` je třída `RuntimeException`, která je využívána pro reprezentaci nejrůznějších běžných typů chyb vzniklých za běhu programu. [9]

Zpracování výjimek v Javě je řízeno pomocí pěti klíčových slov: `try`, `catch`, `throw`, `throws` a `finally`. Těmi je tvořen vzájemně související subsystém, v němž použití jednoho zahrnuje použití druhého. Příkazy programu, u nichž je požadováno sledování výjimek, jsou umístěny v bloku `try`. Dojde – li v bloku `try` k výjimce, pak je tato výjimka vyvolána. V následujícím kroku může být pomocí bloku `catch` zachycen a požadovaným způsobem zpracována. Systémem generované výjimky jsou automaticky vyvolány běhovým systémem Javy. Pro ruční vyvolání výjimky je využíváno klíčového slova `throw`, v některých případech je nutné uvést pomocí klauzule `throws` výjimku, která má být z metody vyvolána. Jakýkoliv kód, který musí být vždy spuštěn po opuštění bloku `try`, je umístěn v bloku `finally`. [9]

Základ pro zpracování výjimek je tvořen klíčovými slovy `try` a `catch`. Tato klíčová slova musí být uvedena společně: v programu nemůže být samostatně uveden příkaz `catch` bez svého `try`. Obecný tvar bloků `try/catch` pro zpracování výjimek (včetně nepovinného bloku `finally`) se uvádí v této podobě:

```
try {
 příkazy; // u nichž se monitoruje chyba
}
catch (typVyjimky vyjimka1) {
 příkazy; // obsluha vyjimky1
}
catch (typVyjimky vyjimka2) {
 příkazy; // obsluha vyjimky2
}
finally {
 příkazy; // finální kód
}
```

Blok `finally` je proveden vždy, když provádění opustí blok `try/catch`, bez ohledu na to, jakými podmínkami bylo toto způsobeno. To znamená, že bez ohledu na to, zda blok `try` skončil normální způsobem nebo kvůli výjimce, je poslední provedený kód definovaný v bloku `finally`. [9]

V Javě je definována uvnitř standardního balíčku `java.lang` sada několika tříd výjimek. Některé často se vyskytující jsou uvedeny v následující tabulce.

Tabulka 2.2 - Nekontrolované výjimky v balíčku java.lang

Výjimka	Popis
`ArithmeticException`	Aritmetická chyba, např. celočíselné dělení nulou.
`ArrayIndexOutOfBoundsException`	Index pole je mimo hranice.
`IllegalArgumentException`	Použití neplatného argumentu pro vyvolání metody.
`NullPointerException`	Neplatné použití prázdné reference.
`NumberFormatException`	Neplatný převod řetězce do číselného formátu.
`StringIndexOutOfBoundsException`	Pokus o indexaci mimo hranice řetězce.
`UnsupportedOperationException`	Pokus o provedení nepodporované operace.

2.2.7 Vstupně – výstupní operace

Vstupně – výstupní operace jsou v jazyku Java založeny na proudech. V moderních verzích Javy jsou definovány dva typy proudů: bajtový a znakový. Bajtové proudy jsou navrženy jako prostředek pro pohodlné zpracování vstupu a výstupu bajtů. Jsou využívány například při čtení či zapisování binárních dat. Užitečné jsou zvláště při práci se soubory. Znakové proudy jsou navrženy pro zpracování vstupu a výstupu znaků, využívají znakovou sadu *Unicode* a mohou být proto internacionalizované. V některých případech je navíc jejich použití efektivnější než použití proudů bajtových. [9]

Bajtové proudy jsou definovány pomocí dvou hierarchií tříd. Na jejich vrcholu jsou dvě abstraktní třídy: `InputStream` a `OutputStream`. Pomocí třídy `InputStream` jsou definovány charakteristiky společné pro bajtové vstupní proudy a pomocí třídy `OutputStream` je popsáno chování bajtových výstupních proudů. Od těchto tříd je dále

odvozeno několik konkrétních podtříd, kterými je nabízena rozličná funkčnost při zpracování detailů čtení a zapisování do nejrůznějších zařízení, jako jsou například diskové soubory. Nejčastěji využívanou podtřídou je pravděpodobně třída `FileInputStream` resp. `FileOutputStream`, sloužící ke čtení ze souboru resp. zápisu do souboru. [9]

Znakové proudy jsou analogicky k bajtovým proudům také definovány pomocí dvou hierarchií tříd, na jejichž vrcholu se nachází dvě abstraktní třídy: `Reader` a `Writer`. Třída `Reader` je využívána pro vstup a třída `Writer` pro výstup. Konkrétní třídy odvozené od tříd `Reader` nebo `Writer` pracují na znakových proudech znakové sady *Unicode*. I od tříd `Reader` a `Writer` je odvozeno několik podtříd, jejichž pomocí lze řešit nejrůznější vstupně – výstupní operace. Často používány jsou podtřídy `BufferedReader/BufferedWriter` (vstupní a výstupní znakový proud s mezipamětí), `FileReader/FileWriter` (vstupní a výstupní proud, sloužící pro čtení/zápis do souboru) a další. [9]

2.3 Základní prvky Java Enterprise Edition

Enterprise edice Java platformy (Java EE) je nástroj pro vývoj business aplikací a informačních systémů. Hlavním cílem Java EE platformy je poskytovat vývojářům výkonný set API rozhraní pro co nejkratší dobu vývoje aplikace, redukci aplikační složitosti a zvýšení aplikačního výkonu. Platforma Java EE je vyvíjena v rámci Java Community Processu (JCP), který je zodpovědný za všechny Java technologie. Platformou Java EE je využíván zjednodušený programovací model, nasazení XML popisovačů je volitelnou možností. Místo tohoto postupu mohou vývojáři jednoduše zadat informaci jako *anotaci* přímo do Java zdrojového souboru, přičemž komponenta je Java EE serverem konfigurována ve chvíli nasazení a běhu. V Java EE platformě mohou být také aplikovány tzv. „injekce závislosti“ (*dependency injections*) ke všem zdrojům, které komponenta potřebuje, přičemž je efektivně oddělena a skryta logika tvorby a vyhledávání ve zdrojích od aplikačního kódu. Injekce závislosti mohou být použity v EJB kontejnerech, webových kontejnerech a aplikačních klientech. Injekce závislosti umožňují Java EE kontejneru automaticky vkládat odkazy na ostatní požadované komponenty nebo zdroje s využitím anotací. [4]

Tabulka 2.3 - Vybrané technologie Java EE

Technologie	Popis
JSP (*JavaServer Pages*)	Technologie, umožňující vkládat speciální direktivu do HTML kódu.

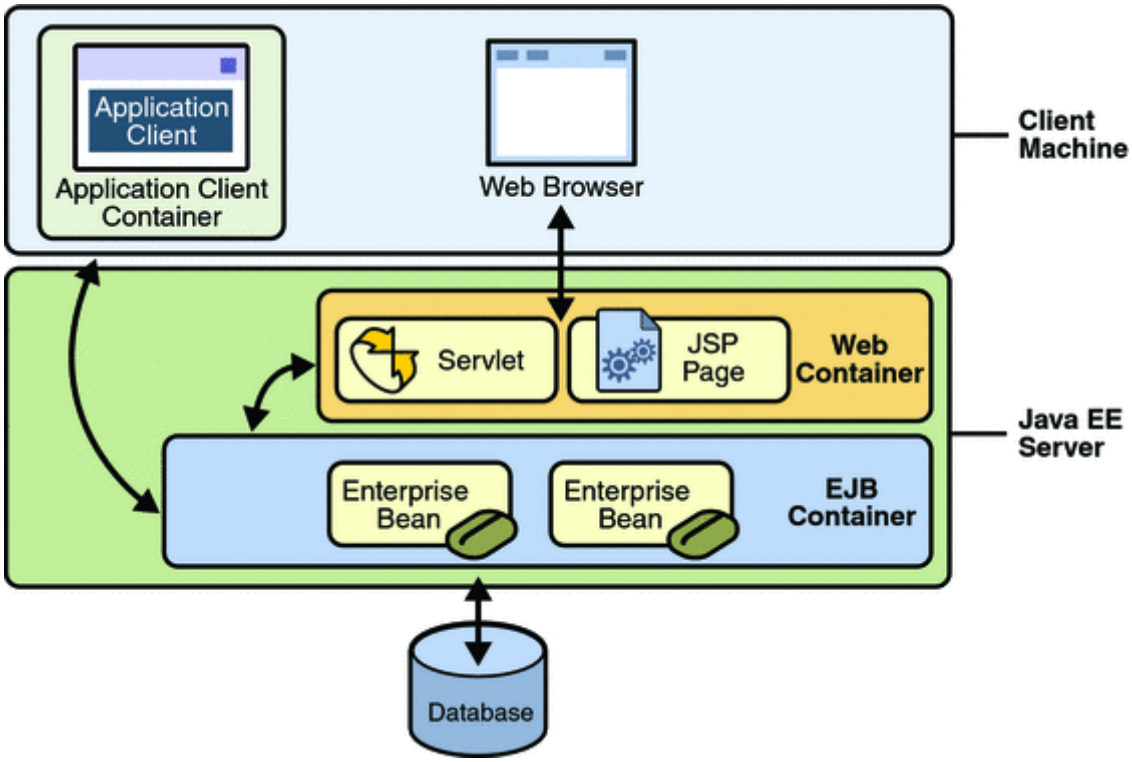
JSF (*JavaServer Faces*)	Konkurenční technologie k JSP. Stránka je reprezentována jako XML soubor.
JDBC (*Java Database Connectivity*)	Standardní rozhraní pro práci s různými typy databází v jazyce SQL.
JPA (*Java Persistence API*)	Rozhraní pro objektovou práci s daty. Konkrétní implementaci reprezentuje Hibernate.
EJB (*Enterprise Java Beans*)	Komponenty pro tvorbu podnikových aplikací.

V tabulce 2.9 jsou uvedeny vybrané základní technologie, které jsou platformou Java EE nabízeny. Některé z nich jsou navzájem konkurenční a záleží pak především na vývojářích, pro použití které dané technologie se nakonec rozhodnou. [10]

2.3.1 Java EE kontejnery

Běžně je obtížné napsat více-úrovňové aplikace pro tenkého klienta, protože to vyžaduje mnoho řádků složitého kódu k obsluze transakcí a stavů, multithreadingu, zdrojových poolů a dalších komplexních nízko-úrovňových detailů. Java EE architektura založená na bázi komponent a platformové nezávislosti je snadno použitelná, neboť business logika je organizována do znovupoužitelných komponent, a navíc Java EE server poskytuje základní služby ve formě kontejneru pro každý typ těchto komponent. Programátorům je tak ušetřen čas, protože nejsou nuceni vyvíjet tyto služby vlastními silami, a mohou se koncentrovat pouze na řešení business problému zadané aplikace. [4]

Kontejnery tvoří rozhraní mezi komponentou a nízko-úrovňovou specifickou funkcionalitou platformy, která komponentu podporuje. Předtím, než je vykonána, musí být webová a aplikační komponenta nebo komponenta enterprise beanu shromážděna v Java EE modulu a umístěna (*deploy*) ve svém kontejneru. Shromažďovací proces zahrnuje specifické nastavení kontejneru pro každou komponentu v Java EE aplikaci a také pro Java EE aplikaci samotnou. Nastavení kontejneru je přizpůsobeno základní podpoře, poskytované Java EE serverem, která zahrnuje služby jako např. zajištění bezpečnosti, transakční management, rozhraní pro Java adresářové a jmenné služby (Java Naming and Directory Interface – JNDI) nebo vzdálenou konektivitu. Úkolem kontejneru je také spravovat nekonfigurovatelné služby, jako např. enterprise beany a životní cykly servletu, zdroje databázové konektivity poolů, perzistenci dat a přístup do API rozhraní Java EE platformy. [4]



Obrázek 2.1 - Kontejnery platformy Java EE (zdroj: docs.oracle.com)

Na výše uvedeném obrázku je zobrazen proces umístění jednotlivých komponent Java EE aplikace v příslušných kontejnerech.

Java EE Server: Běhová (runtime) část produktu Java EE. Java EE server poskytuje EJB a webové kontejnery.

Enterprise JavaBeans (EJB) container: Kontejner, jenž spravuje prováděcí akce enterprise beanů pro Java EE aplikace. Enterprise beany a jejich kontejner běží na Java EE serveru.

Web container: Kontejner, jehož úkolem je spravovat provedení webových stránek, servletů a dalších EJB komponent pro Java EE aplikace. Webové komponenty a jejich kontejner běží na Java EE serveru.

Application client container: Kontejner, v němž jsou spravovány a vykonávány komponenty aplikace klienta. Aplikační klienti a jejich kontejnery běží na straně klienta.

Applet container: V tomto kontejneru je spravována a prováděna funkcionalita apletů. Je složen z webového prohlížeče a Java Plug-inu, běžícím společně na straně klienta. [4]

2.3.2 Technologie Enterprise JavaBean

Komponenta Enterprise JavaBean (EJB) je tělo kódu, obsahující pole a metody pro implementaci modulů business logiky. O enterprise beanu je možné v podstatě uvažovat jako o stavebním bloku, který může být využit samostatně nebo případně s dalšími enterprise beany k vykonání business logiky na Java EE serveru. Enterprise beany mohou být rozděleny na:

- session beany (reprezentující přechodnou komunikaci s klientem – po skončení klientské relace jsou session bean a jeho data ukončeny),
- message-driven beany (kombinující vlastnosti session beanů a message listeneru, umožňující business komponentě přijímat zprávy asynchronně). [4]

2.3.3 Technologie Java Servletů

Servlet je klíčovým stavebním prvkem webové Java aplikace. Technologie Java Servletu umožňuje definovat specifickou HTTP servletovou třídu. Servletová třída rozšiřuje možnosti serverů, který hostí aplikace založené na programovacím modelu typu požadavek – odpověď (request-response). Ačkoliv servlety mohou odpovídat na jakýkoliv typ požadavku, jsou také běžně využívány k rozšíření aplikací poskytovaných webovými servery. [4]

Základní struktura servletu může být popsána asi takto:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Servlet extends HttpServlet {
      public void doGet (HttpServletRequest request, HttpServletResponse
            response)throws ServletException, IOException {
                  příkazy;
      }
     public void doPost (HttpServletRequest request, HttpServletResponse
            response) throws ServletException, IOException {
                  příkazy;
      }
}
```

Servlet musí povinně obsahovat jednu nebo více metod, pomocí kterých je obsluhován příchozí HTTP dotaz, v drtivé většině se jedná o metody GET a POST. Mezi těmito metodami

je rozdíl ve způsobu zasílání dat – zatímco u metody GET jsou data přímo součástí URL adresy (což představuje v některých případech riziko), u metody POST jsou data zasílána v těle HTTP dotazu. Běžně je uvnitř servletu obsažena metoda `processRequest`, ve které jsou uvedeny příkazy, jenž má servlet provádět, a která je volána z příslušné metody `doGet` nebo `doPost`. Mapování servletu je prováděno v XML popisovači (*deployment descriptoru*), jehož úkolem je mj. také ukládání globálních proměnných, nastavení bezpečnosti a autentizace atd. Životní cyklus servletu je ovládán kontejnerem a lze jej rozčlenit do několika kroků – přes načtení třídy servletu, vytvoření instance servletu a volání metody `init()`, až po provádění metody `service()` a volání metody `destroy()` při ukončení celého cyklu. [12]

2.3.4 Technologie JavaServer Pages

Technologie JavaServer Pages (JSP) umožňuje vkládat úryvky servletového kódu přímo do textově-založeného dokumentu. JSP stránka je textově-založený dokument, který obsahuje dva typy textu:

- *statická data*, která mohou být vyjádřena v jakémkoliv textovém formátu jako např. HTML nebo XML,
- *JSP elementy*, které determinují, jak je stránkou konstruován dynamický obsah. [4]

JavaServer Pages představují určitou „nadstavbu“ servletů a jsou využívány pro tvorbu HTML kódu, jenž je prokládán výstupem aplikace. V JSP lze využít několika možností, jak generovat dynamický obsah: skriplety a značky. [13]

Skriplety

JSP je možné prokládat klasickým Java kódem, tvz. skriplety, k jejichž užití jsou deklarovány značky `<% %>`. Skriplety jsou stále podporovány z důvodu jejich výskytu ve starších aplikacích. [13]

Značky

Role skripletů je elegantně nahrazena řadou značek (tagů), jež lze využít takřka na veškeré úkony, které se mohou v praktické aplikaci vyskytnout. Pomocí tagů je obstaráván výpis proměnných, cyklení, formátování textu, vkládání externích šablon, práce s JavaBeany atd. Vývojářům je také umožněno vytvářet si vlastní knihovny tagů a pokrýt tak i specifické

nároky aplikace. V technologii JSP je implicitně poskytováno několik značek, pro jejichž použití není třeba využívat žádné knihovny. Jedná se např. o:

```
<jsp:forward  page="stranka.jsp" />
<!-- přesměrování na další stránku -->

<jsp:include page="stranka.jsp" />
<!-- vložení externí stránky -->

<jsp:useBean id="idBeanu" scope="page/request/session/application"
       class="trida" />
<!-- vložení JavaBeanu, parametr scope udává rozsah platnosti -->

<jsp:getProperty name="idBeanu" property="atribut" />
<!-- získání hodnoty z JavaBeanu -->
```

Samotné JSP tagy ovšem mnoho funkcionality neposkytují. Pro účely větvení kódu, cyklení, vkládání proměnných, formátování řetězců a dalších činností jsou využívány knihovny tagů JSTL (JSP Standard Tag Library). Tyto knihovny je potřeba do aplikace přidat (v podobě JAR souboru – knihovny) a v JSP je nutné vložit direktivu `taglib`, která jejich využití specifikuje. Nejčastěji využívané jsou knihovny `Core` (s prefixem „c“) a `Formatting` (prefix „fmt“). Zvláště knihovna `Core` pak tvoří základ pro definování dynamického obsahu, který bude stránkou JSP generován. [13]

Direktiva `taglib`, pomocí níž je použití knihovny JSTL v JSP stránce umožněno, a také některé základní funkce jsou zobrazeny v následující tabulce.

Tabulka 2.4 - Vybrané značky knihovny JSTL

Standardní syntaxe JSTL	
`<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`	
Značka JSTL Core	**Význam značky**
`<c:out>`	Slouží pro bezpečné vypisování řetězců a proměnných.
`<c:if>`	Jednoduchá podmínka, která obsahuje tělo v případě, že je dodaný výraz v parametru `test` pravdivý.
`<c:choose>`	Startovní tag jednoduché podmínky, který vytváří kontext pro

	tagy <c:when> a <c:otherwise>.
<c:when>	Subtag <c:choose>, který obsahuje tělo, jestliže je dodaná podmínka rovna true.
<c:otherwise>	Subtag <c:choose>, který proběhne pouze tehdy, pokud jsou všechny předešlé podmínky vyhodnoceny jako false.
<c:redirect>	Přesměrování na novou URL adresu.
<c:forEach>	Základní tag pro tvorbu cyklu, akceptující mnoho typů různých kolekcí, podporující podmnožiny a další funkcionality.

Značky knihovny JSTL jsou párové, z čehož vyplývá, že musí být uveden také jejich konec. Ten je tvořen přidáním znaku „/“ do zápisu tagu, stejně jako u jazyka HTML. Takže např. pro značku <c:if> je koncovým tagem </c:if>, pro značku <c:forEach> pak </c:forEach> atd. [16]

2.4 Základní teorie relačních databází

Databáze je chápána jako úložiště údajů, které jsou uloženy a zpracovávány nezávisle na aplikačních programech. Pod pojmem systém řízení báze dat (SŘBD; anglicky Database Management System – DBMS) se rozumí specializovaný serverový software pro přístup k údajům v databázi. Uživateli ani aplikačnímu programu přitom nemusí být známa fyzická struktura uložených údajů, protože k údajům v databázi je přistupováno právě prostřednictvím systému řízení báze dat. Komunikace klienta nebo aplikačního programu se SŘBD je realizován prostřednictvím jazyka SQL. [6]

2.4.1 Základní datové typy v databázi SQL

Každý údaj v databázi je určitého datového typu. V databázových tabulkách jsou údaje uspořádány tak, že každý sloupec obsahuje údaje stejného datového typu. Může to být textový řetězec, číslo, datum, čas atd. [6]

Hodnota NULL, která je do jisté míry společná všem datovým typům, vyjadřuje neexistenci hodnoty proměnné. Je ale třeba přísně rozlišovat mezi nulovou hodnotou proměnné nebo prázdným řetězcem a hodnotou NULL, neboť nula je v podstatě konkrétní číslo, zatímco hodnota NULL znamená, že údaj nebyl zadán a že není znám. [6]

V následující tabulce jsou uvedeny a popsány základní datové typy jazyka SQL, které jsou využívány také v prostředí Microsoft SQL Serveru.

Tabulka 2.5 - Datové typy Transact-SQL pro Microsoft SQL Server

Datové typy Microsoft SQL Serveru (Transact-SQL)			
Název typu	Popis	Rozsah/délka	Paměť
Znakové datové typy			
`char`	Textový řetězec s pevnou délkou, určenou v závorce.	(n; 1 - 8000)	n bytů
`varchar`	Textový řetězec s proměnnou délkou, určenou v závorce.	(n; 1 - 8000), MAX 2^31-1	n bytů
`text`	Text s proměnnou délkou.	MAX 2^31-1	n bytů
Přesné číselné datové typy			
`tinyint`	Celé číslo.	0 až 255	1 byte
`smallint`	Celé číslo.	-2^15 až 2^15-1	2 byty
`int`	Celé číslo.	-2^31 až 2^31-1	4 byty
`bigint`	Dlouhé celé číslo.	-2^63 až 2^63-1	8 bytů
`decimal`	Číslo s pevnou desetinnou čárkou.	-10^38+1 až 10^38-1	5-17 bytů
Aproximované číselné datové typy			
`float`	Číslo s pohyblivou desetinnou čárkou.	-1,79E+308 až 1,79E+308	4 nebo 8 bytů
`real`	Číslo s pohyblivou desetinnou čárkou.	-3,40E+38 až 3,40E+38	4 byty
Datové typy pro datum a čas			
`date`	Definuje datum.	YYYY-MM-DD	3 byty
`datetime`	Definuje datum a čas s přesností na setiny sekundy ve 24 hodinovém formátu.	YYYY-MM-DD hh:mm:ss[.fff]	8 bytů

Kromě uvedených základních datových typů existují samozřejmě další - v číselných typech např. `BIT` pro vyjádření 1 nebo 0, `SMALLMONEY` a `MONEY` pro vyjádření měny, v binárních řetězcích pak typy `BINARY`, `VARBINARY` či `IMAGE`. Velmi často je také využíván datový typ `TIMESTAMP`, časové razítko. [15]

2.4.2 Základní struktura jazyka SQL

Jazyk SQL (Structured Query Language) je standardizovaný dotazovací jazyk, jenž je využíván pro práci s daty v relačních databázích. Systematická kategorizace rozděluje příkazy jazyka SQL na několik základních podmnožin:

- *Data Definition Language* (DDL, česky Jazyk pro definici dat),
- *Data Manipulation Language* (DML, česky Jazyk pro manipulaci s daty),
- *Data Control Language* (DCL, česky Příkazy pro řízení přístupu),
- *Transaction Control Commands* (TCC, česky Příkazy pro řízení transakcí). [6]

Data Definition Language

Pomocí příkazů z množiny DDL je umožněno definovat struktury a vytvářet v databázi různé objekty, jako například tabulky, pohledy, indexy atd. Těmito příkazy lze také měnit strukturu těchto objektů nebo je rušit. Patří sem příkazy jako `CREATE DATABASE`, `CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`, `CREATE INDEX`, `DROP INDEX`, `CREATE VIEW`, `ALTER VIEW` a další. [6]

Data Manipulation Language

Pomocí příkazů tohoto jazyka je umožněna manipulace s údaji, to znamená výběr a vkládání údajů a jejich aktualizace, dále také mazání (odstraňování) údajů a samozřejmě také velmi mocný příkaz `SELECT` pro výběr údajů. Základní příkazy, které patří do DML jsou `SELECT`, `INSERT`, `UPDATE` a `DELETE`. [6]

Data Control Language

V DCL je obsažena podmnožina řídících příkazů se speciálními příkazy pro řízení provozu a údržbu databáze. Důležitými příkazy jsou také ty, jimiž je možné přidávat a odebírat uživatelská přístupová práva pro jednotlivé uživatele a skupiny uživatelů. Do této skupiny lze zařadit například následující příkazy: `CREATE USER`, `ALTER USER`, `DROP USER`, `GRANT`, `REVOKE` atd. [6]

2.4.3 Vybrané příkazy jazyka SQL

Jazyk SQL samozřejmě obsahuje spoustu rozličných příkazů, pomocí kterých je vytvářena struktura databáze, manipulováno s daty atd. Pro vytvoření struktury databázových tabulek je využíváno příkazu `CREATE TABLE`, jehož obecná podoba je následující:

```
CREATE TABLE [schema].jmeno_tabulky
(
 jmeno_sloupce datovy_typ [DEFAULT vyraz],
 jmeno_sloupce2 datovy_typ [DEFAULT vyraz],
 ...
 jmeno_sloupceN datovy_typ [DEFAULT vyraz]
)
```

Volbou `DEFAULT` je specifikována implicitní hodnota, která je využita při vkládání nového řádku, pokud by pro daný sloupec nebyla uvedena konkrétní hodnota. Pro zabránění zadávání nesprávných hodnot do databázových tabulek je v některých případech vhodné zavést na některé sloupce určitá omezení. Všechna omezení lze přidávat i odebírat, nelze je však upravovat. Přidání a odebrání daného konkrétního omezení je realizováno prostřednictví příkazu `ALTER TABLE`. Jako příklady omezení lze uvést `FOREIGN KEY` (cizí klíč v jiné tabulce), `CHECK` (definice podmínek a omezení, které musí být splněny pro každý záznam), `NOT NULL` (zabránění vložení hodnoty `NULL` - hodnota sloupce musí být vždy specifikována), `PRIMARY KEY` (primární klíč, jednoznačný identifikátor, který musí být vždy uveden) nebo `UNIQUE` (zajišťuje unikátní hodnotu sloupce pro všechny záznamy v tabulce). [6]

Jednoznačně nejvyužívanějším příkazem jazyka SQL je dotazovací příkaz SELECT. Jeho základní syntaxi lze uvést v následující podobě:

```
SELECT [*] [seznam_polozek_vystupni_sestavy]
FROM jmeno_tabulky
WHERE podminka_vyberu
GROUP BY polozky
HAVING podminka_agregace
ORDER BY seznam_polozek [ASC] [DESC]
```

Hvězdička za příkazem `SELECT` je zástupný znak, indikující výběr všech sloupců databázové tabulky. V dotazech typu `SELECT` lze také využívat nepovinnou klauzuli `AS`, sloužící pro

přiřazení nových názvů sloupců (*aliasů*), dále různé varianty klauzule `JOIN` (konkrétně `CROSS JOIN` pro křížové spojení, `FULL JOIN` pro úplné spojení, `INNER JOIN` pro vnitřní spojení, `LEFT OUTER JOIN` pro vnější spojení z levé strany a `RIGHT OUTER JOIN` pro vnější spojení z pravé strany). Klauzule `WHERE` specifikuje konkrétní podmínku pro výběr, klauzule `GROUP BY` je využívána při agregaci dat, stejně jako `HAVING`, kde tato část dotazu představuje podmínku pro agregovaná data. Pomocí klauzule `ORDER BY` lze pak vybraná data seřadit vzestupně (`ASC`) nebo sestupně (`DESC`). [6]

2.5 Formát PGN datového souboru

PGN (Portable Game Notation, česky formát přenosné herní notace) je standardní formát pro reprezentaci dat šachových partií, využívající ASCII textové soubory. Formát PGN je strukturován pro jednoduché čtení a zapisování uživatelem, stejně jako pro snadné rozkódování a vytváření za pomoci šachových programů. [11]

PGN je celosvětově nejrozšířenější formát používaný pro ukládání, přenos a následnou rekonstrukci šachových partií, jeho výhody spočívají především v jednoduchosti a snadné čitelnosti (soubory s příponou PGN lze běžně otevřít např. v poznámkovém bloku a vyčíst z nich potřebné informace). Práci se soubory typu PGN podporují veškeré šachové softwary předních světových firem, zabývající se touto oblastí (např. německá ChessBase a její produkty). Nevýhody spočívají především ve špatném kódování znaků, kvůli čemuž je i šachovým svazem ČR a dalšími organizacemi vydáno doporučení, nepoužívat při tvorbě PGN souborů diakritiku.

2.5.1 Specifikace formátu PGN

PGN soubor je tvořen takzvanou „sekcí párových tagů“. Párový tag se skládá ze čtyř po sobě jdoucích znaků (tokenů) − levé závorky, symbolického znaku, znaku řetězce a pravé závorky. Symbolickým znakem je jméno tagu a znakem řetězce je pak konkrétní hodnota spojená s tímto jménem. Stejné jméno tagu se nesmí vyskytnout více než jednou v sekci párových tagů. Definice formátu PGN určuje, že se mezi levou závorkou a jménem tagu nesmí objevit mezera, stejně jako se nesmí vyskytnout mezi pravou závorkou a znakem řetězce. Mezi jménem tagu a znakem řetězce (hodnotou tagu) musí být vložena právě jedna mezera. Jména tagů, stejně jako všechny symboly, jsou case-sensitive (je rozlišováno mezi malými a velkými písmeny). Zápis jednotlivých tahů musí být oddělen mezerami a na konci se musí vyskytnout označení výsledku partie. [11]

V PGN je skupina tagů, které je povinné uvést pro archivní uložení PGN formátu. Tato skupina je označena jako „*seznam sedmi tagů*“ (Seven Tag Roster, STR). Interpretace těchto tagů je fixní, stejně jako pořadí, ve kterém se musí v PGN souboru objevit. Těchto sedm povinných tagů je tvořeno následujícími položkami (seřazeno dle pořadí):

- *Event* (událost) − název turnaje nebo zápasu,
- *Site* (místo) − místo dané události,
- *Date* (datum) − startovní datum dané partie,
- *Round* (kolo) − hrací kolo dané partie,
- *White* (bílý) − hráč, hrající bílými kameny v partii,
- *Black* (černý) − hráč, hrající černými kameny v partii,
- *Result* (výsledek) − výsledek partie.

Kromě těchto tagů bývají často uváděny také další nepovinné − *WhiteElo* (elo koeficient bílého), *BlackElo* (elo koeficient černého) a *ECO* (kód zahájení dle knihovny zahájení). Existují i další tagy, které jsou ovšem využívány méně, např. *TimeControl* (časové tempo na partii), *WhiteTeam* (tým bílého), *BlackTeam* (tým černého) apod. [11]

2.5.2 Konkrétní implementace formátu PGN

Konkrétní implementace formátu PGN musí dodržovat výše uvedené omezení a pro jednotlivé znaky řetězců jsou samozřejmě určeny další omezení nebo vzory. Vzorový zápis partie ve formátu PGN může být zobrazen asi takto:

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3
d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7 11. c4 c6 12. cxb5 axb5 13. Nc3
Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5 Nxe4 18. Bxe7 Qxe7 19.
exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6 23. Ne5 Rae8 24.
Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5 hxg5 29.
b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5
35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6
Kc5 41. Ra6 Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

Položky *Event*, *Site*, *White* a *Black* jsou čistě textového charakteru, mohou obsahovat mezery, tečky, dvojtečky, pomlčky a další znaky. Položka *Date* je definována ve standardním formátu „`YYYY.MM.DD`“, s tím, že pokud není některý údaj znám, je nahrazen znakem „`?`“. Položka *result* musí být reprezentována jednou z hodnot `1-0`, `0-1`, `1/2-1/2` nebo `*`. Jiné možnosti zde nejsou přípustné. [11]

3 Návrh a realizace SQL databáze a aplikační logiky v jazyce JAVA

Tato kapitola je svým obsahem zaměřena na popis návrhu datové struktury v jazyce SQL, návrhu aplikační logiky v jazyce Java a následně na popis samotné realizace tohoto návrhu. V úvodní části jsou formou use-case diagramů vysvětleny základní případy užití aplikace v praxi. Další část je cílena na popis relačního modelu databáze, včetně jednotlivých entit této databáze, a stručného nastínění jejich smyslu a obsahu. V poslední části je podrobně popsána aplikační logika v jazyce Java a konkrétní využití jednotlivých technologií, popsaných v teoretických východiscích práce.

3.1 Use-case diagram systému

Diagram případů užití (use-case diagram) je využíván pro popis rozsahu systému, vymezení jednotlivých případů užití (těmi jsou myšleny jednotlivé akce příp. události, které mohou v systému nastat) s vazbami na příslušné uživatele (aktéry) systému.

3.1.1 Kandidáti na aktéra a případy užití

V následující tabulce jsou zachyceny návrhy kandidátů na aktéra systému a možné případy užití, připadající k danému kandidátovi v úvahu.

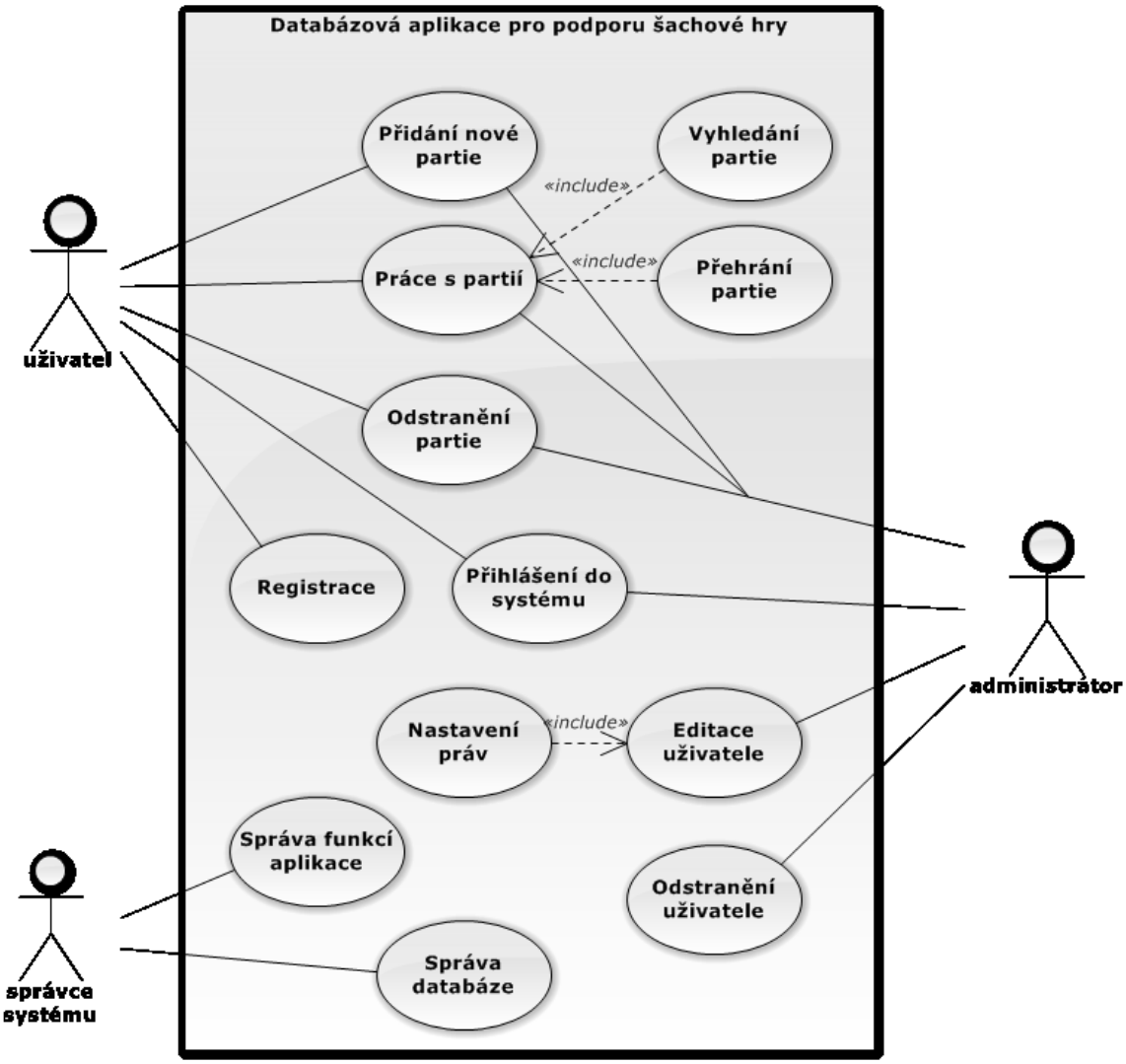
Tabulka 3.1 - Tabulka kandidátů a jejich případů užití

Tabulka kandidátů	
Aktéři (podstatná jména)	Use Case (slovesa + předmět)
Administrátor	Přihlašuje se
	Přidává uživatele
	Spravuje uživatele
	Odstraňuje uživatele
Uživatel	Registruje se
	Přihlašuje se
Správce systému	Spravuje funkce aplikace
	Spravuje databázi
Partie	Je přidávána
	Je editována
	Je vyhledávána

	Je přehrávána
	Je odstraněna

3.1.2 Grafické zobrazení Use-case diagramu

Na následujícím obrázku je zobrazen pohled na celý systém databázové aplikace pro podporu šachové hry, zahrnující jednotlivé případy užití a jejich spojení s danými aktéry, kterých se konkrétní případ užití přímo týká.



Obrázek 3.1 - Diagram případů užití databázové aplikace pro podporu šachové hry

Do systému vstupují tři hlavní aktéři (uživatel, administrátor a správce systému), kteří svými akcemi mění jednotlivé entity systému. Pro lepší pochopení fungování jejich vzájemné

interakce se systémem jsou zde uvedeny uživatelské požadavky a pravděpodobné profily konkrétních aktérů.

3.1.3 Katalog uživatelských požadavků

V následující tabulce jsou zobrazeny nejběžnější uživatelské požadavky jednotlivých aktérů. Tyto požadavky jsou důležité pro následný popis možných scénářů případů užití.

Tabulka 3.2 - Katalog uživatelských požadavků

Č. pož.	Aktér	Požadavek	Priorita
P1	Uživatel	Registrace do systému	1
P2	Uživatel	Přehrání partie	1
P3	Uživatel	Přidání partie	2
P4	Administrátor	Editace uživatele	2
P5	Administrátor	Odstranění uživatele	1
P6	Administrátor	Přidání partie	1
P7	Správce systému	Přidání nové funkce aplikace	2

3.1.4 Tabulka profilů aktérů

Tabulka profilů aktérů je určena k získání úsudku a zběžné představě o osobnostních profilech konkrétních aktérů v systému. Především jsou v ní obsaženy informace o schopnostech a dovednostech aktéra, může také obsahovat i některé další doplňující informace a parametry jednotlivých uživatelských rolí.

Tabulka 3.3 - Tabulka profilů aktérů s osobnostními profily

Aktér	Profil: Kvalifikace a schopnosti
Uživatel	Osoba se zájmem o šachovou hru a zběžně znalá základní práce s počítačem. Zvídavá osobnost, která bere šachovou partii velice vážně a neváhá obětovat přípravě a úspěchu v této hře nemalou porci svého volného času. Inteligentní, může být mírně introvertní, rozvážný, klidný, trpělivý.
Administrátor	Zkušená a zodpovědná osoba, schopná dohlížet na dodržování pravidel, s výbornou znalostí systému a všech jeho funkcí. Pracovitá osobnost, pomáhající zlepšovat systém, udržovat pořádek v jeho

	jednotlivých částech. Zajišťování chodu systému bere jako koníček, zábavu, snaží se rozšiřovat možnosti nabízené běžným uživatelům. Vyniká asertivním chováním, nenechá se vyvézt z míry, má velmi dobré komunikativní schopnosti, rozumí základním funkčním principům aplikační logiky systému, sám dokáže vyřešit většinu vzniklých komplikací.
Správce systému	Programátor, dokonale znalý všech funkcí systému. Pracovitý, zodpovědný a pečlivý při tvorbě, testování a přidávání nových funkcí do systému. Velmi dobrý v komunikaci s administrátory, schopný porozumět profesionálním i laickým požadavkům na změnu systému. Rychle pracující, fungující operativně na základě požadavků administrátorů.

3.1.5 Scénáře případů užití

V níže uvedených tabulkách jsou zadokumentovány základní scénáře jednotlivých případů užití. Pomocí těchto tabulek je vždy popsán konkrétní scénář a postup vybraného případu užití při určitých vstupních a výstupních podmínkách, může být také nastaven spouštěč a priorita tohoto případu užití a případně také četnost užití. Tabulka užití č. 1 se týká registrace uživatele (v diagramu je o něm hovořeno jako o „osobě“, neboť v daný okamžik ještě není pravým registrovaným uživatelem systému) a postupnému popisu činností, které je potřeba k úspěšnému provedení této akce vykonat.

Tabulka případu užití č. 1			
Identifikátor:	**UC1**	Název:	**Registrace uživatele**
Aktéři:	Uživatel (osoba)		
Popis:	Osoba se rozhodne stát se uživatelem databázové aplikace pro podporu šachové hry. Vyhledá si příslušný registrační formulář na konkrétní stránce aplikace, korektně vyplní požadované informace a vyčká na potvrzení systému o registraci.		
Vstupní podmínky:	1. Osoba disponuje příslušnými technickými prostředky (PC, připojení k internetu, webový prohlížeč). 2. Osoba zná cestu k příslušné registrační stránce aplikace. 3. Osoba je kvalifikována k vyplnění a odeslání formuláře dle		

	zadaných požadavků aplikace.
Výstupní podmínky:	1. Záznam o uživateli je uložen v databázi aplikace.
Hlavní scénář:	1. Osoba zadá do prohlížeče adresu databázové aplikace pro podporu šachové hry. 2. Osoba vyhledá na zobrazené stránce příslušný odkaz na registrační stránku. 3. Osoba pravdivě a korektně vyplní požadované údaje (*login, heslo, jméno, příjmení, e-mailová adresa*). 4. Osoba odešle formulář pomocí registračního tlačítka a vyčká odpovědi serveru s potvrzením o registraci. 5. Osoba se stává uživatelem systému.
Rozšíření:	1a: Osoba nezná adresu databázové aplikace pro podporu šachové hry. 1a1: Osoba vyhledá adresu databázové aplikace pro podporu šachové hry pomocí vyhledávače nebo kontaktuje osobu, jíž je tato adresa známa. 3a: Osoba vyplní nekorektní údaje v registračním formuláři. 3a1: Osoba je upozorněna pomocí kontrolního mechanismu na nesprávné vyplnění formuláře s označením nekorektních polí. 3a2: Osoba je vyzvána k opětovnému zadání požadovaných údajů. 3b: Osobou zadané uživatelské jméno je již používáno jiným uživatelem. 3b1: Osoba je upozorněna na shodu uživatelského jména s jiným uživatelem pomocí kontrolního mechanismu. 3b2: Osoba je vyzvána k opětovnému zadání rozdílného uživatelského jména. 4a: Aplikace nereaguje na odeslaný uživatelský požadavek. 4a1: Osoba vyčká na odpověď serveru, případně znovu po určité době opakuje svůj požadavek. 4a2: V případě delšího výpadku osoba kontaktuje správce systému nebo administrátora systému na uvedeném kontaktu.
Spouštěč:	Osoba projeví zájem využívat funkce systému.
Priorita:	vysoká

Četnost užití:	jednorázová činnost
Následné podmínky:	Osoba se stává uživatelem systému.

V tabulce užití č. 2 je zachycena hlavní podstata celé databázové aplikace pro podporu šachové hry – přehrání šachové partie. Partie může být uživatelem získána z nahraného souboru ve formátu PGN nebo výběrem z databáze na základě určitého kritéria.

Tabulka případu užití č. 2			
Identifikátor:	**UC2**	Název:	**Přehrání partie**
Aktéři:	Uživatel		
Popis:	Uživatel vybere z nahraného souboru nebo z databázového výběru partii, o jejíž přehrání má zájem. Pomocí akčního tlačítka se pohybuje v zápise partie a sleduje pohyb figur na šachovnici.		
Vstupní podmínky:	1. Uživatel je řádně přihlášen do systému. 2. Uživatel disponuje souborem ve formátu PGN s příslušným korektním zápisem nebo vyhledanou sadou záznamů z databáze. 3. Uživatel chápe význam zápisu jednotlivých tahů, rozumí symbolům na šachovnici a chápe souvislosti v šachové partii.		
Výstupní podmínky:	1. Uživatel získá z přehrání partie určitý užitek v podobě informací.		
Hlavní scénář:	1. Uživatel se přihlásí do systému pomocí svého uživatelského jména a hesla. 2. Uživatel nahraje pomocí formuláře soubor ve formátu PGN, obsahující korektní zápis partií, které požaduje přehrát. 3. Uživatel vybere z rozbalovacího seznamu konkrétní partii, jež má být přehrávačem přehrána. 4. Uživatel pomocí akčního tlačítka prochází jednotlivé tahy partie až do konečné pozice (akční tlačítko neaktivní).		
Rozšíření:	1a: Uživatel zadal neexistující uživatelské jméno nebo neplatné heslo a nemůže se přihlásit do aplikace. 1a1: Uživatel vyplní korektní hodnoty uživatelského jména a hesla a pokusí se o nové přihlášení. 2a: Systém hlásí chybu při nahrávání souboru. 2a1: Uživatel zadá soubor ve formátu PGN o velikosti max. 2MB a		

	pokusí se o nové nahrání souboru. 2b: Uživatel chce přehrát partii umístěnou v databázi. 2b1: Uživatel zvolí stránku pro vyhledávání partií z databáze. 2b2: Uživatel zadá požadovaná kritéria pro výběr partie z databáze. 2b3: Uživatel z rozbalovací lišty se seznamem partií vybere příslušnou partii k přehrání. 3a: Uživateli se nezobrazí rozbalovací lišta se seznamem partií. 3a1: Uživatel nahraje jiný soubor PGN s korektním zápisem partií. 3b: Uživateli se nezobrazí rozbalovací lišta se seznamem partií, vybraných z databáze. 3b1: Uživatel se vrátí na stránku pro výběr partií z databáze a pokusí se svůj dotaz opakovat. 4a: Uživatel se rozhodne partii procházet znova od počátečního tahu. 4a1: Uživatel zvolí akční tlačítko „Původní postavení“ a pokračuje v přehrávání partie opět od 1. tahu.
Spouštěč:	Potřeba přehrát si partii za účelem získání určitého užitku.
Priorita:	vysoká
Četnost užití:	zhruba 1-2x týdně v dávce 5-20 partií
Následné podmínky:	Uživatel získá informaci o hře potencionálního soupeře.

Z administrátorského pohledu je velice důležité udržovat systém v přijatelné podobě. Toto je zde myšleno jako udržitelný nárůst počtu partií v databázi, kontrolu chování jednotlivých uživatelů a případné zablokování nebo odstranění účtu neseriózního uživatele. Samozřejmě takovéto pravomoci musí být svěřeny pouze vysoce pověřené osobě, která nebude mít v úmyslu rozvoji aplikace škodit, ale podílet se na jejím rozvoji a dalšímu expandování. Každopádně v případě hrubého opakovaného porušení pravidel aplikace nemá administrátor jinou možnost než účet viníka smazat, a tím mu zamezit v přístupu k funkcím aplikace. Tato situace je popsána v tabulce případu užití č. 3.

Tabulka případu užití č. 3			
Identifikátor:	**UC3**	Název:	**Odstranění uživatele**
Aktéři:	Administrátor, uživatel		
Popis:	Uživatel hrubým způsobem opakovaně poruší pravidla databázové		

	aplikace pro podporu šachové hry. Administrátor je v rámci pravidel použití aplikace nucen mu za tento prohřešek smazat uživatelský účet a zamezit mu tak v přístupu k uživatelským funkcím této aplikace.
Vstupní podmínky:	1. Uživatel hrubým způsobem a opakovaně poruší pravidla stanovená databázovou aplikací pro podporu šachové hry. 2. Uživatel po výzvě k ukončení chování proti pravidlům aplikace na tyto výzvy nereaguje a zcela vědomě je ignoruje. 3. Administrátor monitoruje chování uživatele, jenž pravidla porušuje.
Výstupní podmínky:	1. Uživatelův účet je smazán z databáze.
Hlavní scénář:	1. Uživatel se dopustí prohřeškům proti pravidlům databázové aplikace pro podporu šachové hry. 2. Administrátor při zběžné kontrole zjistí porušení pravidel aplikace konkrétním uživatelem. 3. Administrátor varuje prostřednictvím e-mailové zprávy uživatele na porušení pravidel a hrozí případným smazáním účtu v případě dalšího trvání této činnosti. 4. Uživatel ignoruje varování administrátora a dále pokračuje v činnosti proti pravidlům aplikace. 5. Administrátor při zběžné kontrole zjistí porušení pravidel aplikace konkrétním uživatelem. Při kontrole uživatele zjistí, že se jedná o opakovaný prohřešek. 6. Administrátor provede odstranění účtu uživatele z databáze uživatelů.
Rozšíření:	2a: Administrátor je upozorněn některým z uživatelů na porušování pravidel aplikace jiným uživatelem. 2a1: Administrátor prověří kontrolou oprávněnost uživatelského podnětu ohledně porušování pravidel jiným uživatelem. 4a: Uživatel uzná svůj prohřešek. 4a1: Uživatel vyrozumí administrátora s vědomím vlastní chyby. 4a2: Uživatel v činnosti proti pravidlům nadále nepokračuje. 5b: Administrátor se rozhodne uživatele opět varovat. 5b1: Administrátor zašle uživateli e-mailovou zprávu s poslední výzvou k ukončení činnosti v rozporu s pravidly aplikace.

Spouštěč:	Zjištění porušení pravidel ze strany uživatele administrátorem.
Priorita:	vysoká
Četnost užití:	jednorázová činnost
Následné podmínky:	Uživatelův účet je úspěšně smazán z databáze uživatelů.

Posledním aktérem systému je správce systému. Toto označení je zde možná zavádějící, ale pod tímto pojmenováním je ukryt specialista – programátor, jenž má na starosti kontrolu funkčnosti a případné přidávání nových funkcí do aplikace, stejně jako kontrolu správné funkčnosti databáze a případnou úpravu její struktury. Jedna z jeho hlavních úloh je specifikována a popsána v tabulce užití č. 4.

Tabulka případu užití č. 4			
Identifikátor:	**UC4**	Název:	**Přidání nové funkcionality**
Aktéři:	Správce systému, administrátor		
Popis:	Administrátor zjistí na základě užívání aplikace chybějící funkcionalitu. Administrátor se obrátí s požadavkem na doplnění funkcionality na správce systému. Správce systému ve vývojovém prostředí chybějící funkcionalitu doplní a uvede do provozu novou verzi aplikace.		
Vstupní podmínky:	1. Administrátor používá aplikace pravidelně a je detailně seznámen s jejím fungováním. 2. Administrátor narazí na chybějící funkcionalitu. 3. Administrátor úzce spolupracuje se správcem systému. 4. Správce systému není zaneprázdněn a je schopen chybějící funkcionalitu vlastními silami do aplikace doplnit.		
Výstupní podmínky:	1. Aplikace je rozšířena o novou funkcionalitu.		
Hlavní scénář:	1. Administrátor zjistí chybějící funkcionalitu, kterou by bylo podle jeho názoru velmi vhodné do aplikace doplnit. 2. Administrátor upozorní na tento nedostatek správce systému. 3. Správce systému ověří hlášení administrátora. 4. Správce systému připraví návrh na doplnění funkcionality, jenž následně konzultuje s administrátorem. 5. Správce systému implementuje návrh na doplnění funkcionality do		

	zdrojového kódu aplikace. 6. Správce systému uvede do provozu novou verzi aplikace s vyšším pořadovým číslem.
Rozšíření:	1a: Administrátor je na chybějící funkcionalitu upozorněn uživatelem. 1a1: Administrátor prověří uživatelský podnět a vyrozumí uživatele o oprávněnosti jeho požadavku. 3a: Správce systému je již s nedostatkem seznámen. 3a1: Správce systému informuje administrátora o aktuálním stavu řešení návrhu a stavu případné implementace požadované funkcionality. 3a2: Správce systému zvýší prioritu provádění výše uvedeného na co nejvyšší. 4a: Správce systému uzná upozornění na chybějící funkcionalitu jako neopodstatněné nebo velmi těžce realizovatelné v praxi. 4a1: Správce systému oznámí svůj postoj administrátorovi. 4a2: Správce systému zahájí diskuzi napříč administrátory s možností přizvat ke konzultaci zkušené uživatele. 5a: Návrh je nemožné implementovat do zdrojového kódu aplikace. 5a1: Správce systému přepracuje návrh. 5a2: Správce systému konzultuje přepracovaný návrh s administrátorem (či administrátory nebo zkušenými uživateli). 5a3: Správce systému implementuje přepracovaný návrh do zdrojového kódu aplikace. 5b: Správce systému je zaneprázdněn jinou činností. 5b1: Správce systému ověří možnosti spolupráce s externím programátorem nebo společností. 5b2: Práce na implementaci nové funkcionality systému je svěřena externí pracovní síle (za předpokladu vysoké priority implementace nové funkcionality a také zvýšených nákladů). 6a: Správce systému neuvede novou verzi aplikace do provozu okamžitě po implementaci nové funkcionality. 6a1: Správce systému přidá více funkcionalit postupně. 6a2: Správce systému uvede do provozu verzi systému s více

	přidanými funkcionalitami.
Spouštěč:	Chybějící funkcionalita aplikace.
Priorita:	středně vysoká
Četnost užití:	1 – 2x do roka
Následné podmínky:	Uvedení vyšší verze aplikace s rozšířenou funkčností.

Kromě těchto čtyř uvedených případů užití lze samozřejmě v systému nalézt také další, které zde nejsou takto podrobně specifikovány a popsány. Za všechny mohou být uvedeny např. operace s partiemi (přidání, vyhledávání, odstranění). Tyto případy užití by pak bylo možné dále rozdělit podle hlavních aktérů, kteří se na jejich provádění podílí.

3.2 Návrh a realizace SQL databáze

Návrh databázové struktury je řešen na základě formálních požadavků, které je možné vyčíst z předpisu formátu PGN. Hlavním úkolem aplikace je uchovávat v databázi jednotlivé parametry každé partie a také informace o uživatelských účtech, pomocí kterých je uživatelům zabezpečen k těmto informacím přístup. Struktura databázových objektů byla navržena tak, aby byl možný výskyt redundance dat omezen na minimum. Návrh struktury musel být proveden samozřejmě velice pečlivě, protože každý pozdější zásah do struktury relační databáze za provozu bývá náročnou záležitostí.

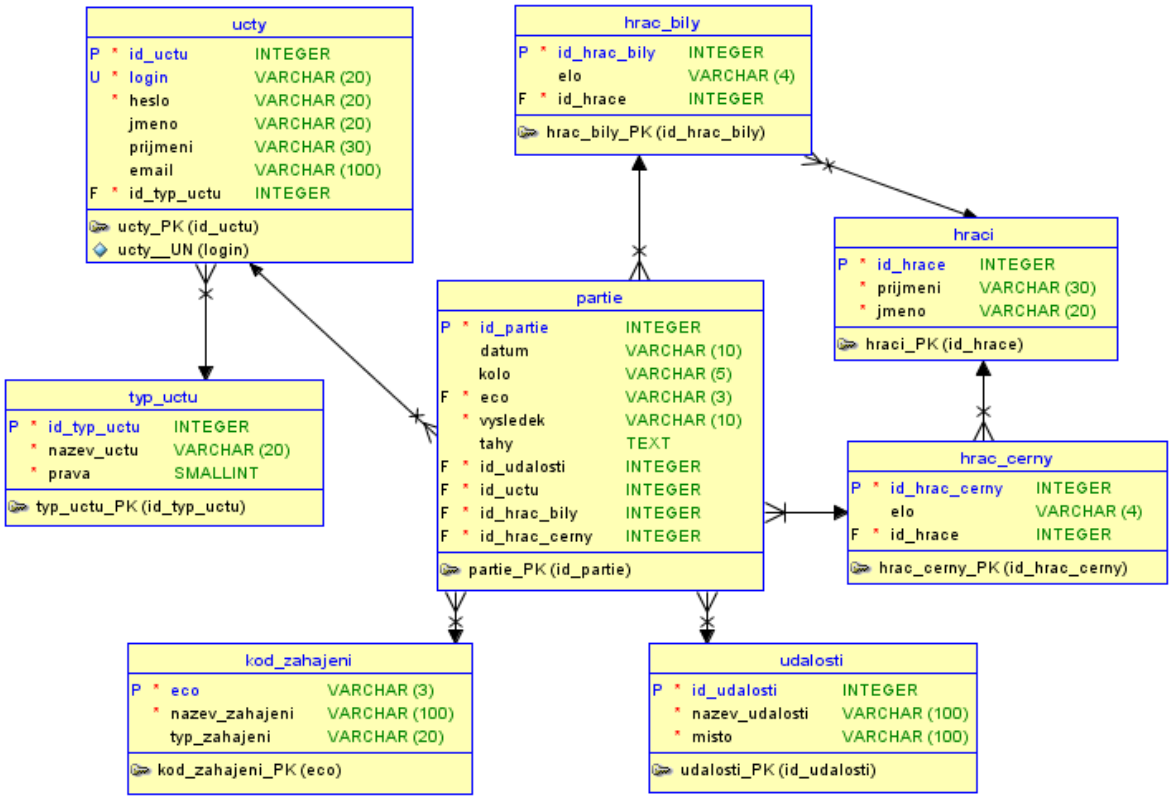
3.2.1 Návrh modelu relační databáze

Pro návrh modelu relační databáze, který byl následně implementován do aplikace pro podporu šachové hry, bylo využito programu Oracle SQL Developer Data Modeler verze 4.0.0.825 (dále jen Modeler). V tomto softwaru je, kromě nástrojů pro modelování na logické i fyzické úrovni, obsažena také podpora pro multi-dimenzionální modelování. Mimo Oracle Database je navíc podporována také databáze IBM DB2, Microsoft SQL a další, k jejichž chodu je využíváno JDBC driverů.

Postup práce v programu Modeler probíhal v následujících krocích:

- vytvoření jednotlivých entit,
- naplnění entit atributy s určením datového typu,
- nastavení primárního klíče každé entity,
- nastavení unikátního klíče u příslušných atributů,
- vytvoření relací mezi entitami pomocí cizích klíčů,
- export DDL souboru pro Microsoft SQL Server.

Relační model, jenž byl v programu Modeler navrhnut pro pozdější realizace v prostředí Microsoft SQL Serveru, je zobrazen na následujícím obrázku.



Obrázek 3.2 - Návrh relačního modelu v programu Oracle SQL Developer Data Modeler

Po vytvoření tohoto relačního modelu byl programem Modelerem vygenerován kód DDL souboru (dotazu), určený pro spuštění nad databází Microsft SQL Serveru. Takto vytvořený dotaz byl dodatečně modifikován (konkrétně přidáním vlastnosti `IDENTITY(1,1)` u všech

číselných primárních klíčů jednotlivých entit) a následně použit v databázi Microsoft SQL Serveru. Pokud by měly být uvedené technické detaily o serveru, jedná se o:

- server IP – 158.196.160.27,
- produkt – Microsoft SQL Server Standart Edition (64-bit),
- verze – 10.0.1600.22,
- operační systém – Microsoft Windows NT 6.0 (6002),
- paměť – 16378 MB, počet procesorů – 8.

3.2.2 Realizace a popis tabulek relační databáze

Provedením dotazu jazyka DDL nad určenou databází byla na serveru vytvořena požadovaná struktura z relačního modelu programu Modeler. Na serveru byly vytvořeny databázové tabulky, obsahující příslušné klíče a indexy. Referenční integrita je zajištěna pomocí primárních a cizích klíčů v daných tabulkách, u tabulek je nastavena vlastnost kaskády při smazání záznamu v závislé tabulce.

Tabulka 3.4 - Tabulka "ucty"

Tabulka „ucty“			
Název atributu	Datový typ	Vlastnosti	Příklady hodnot
id_uctu	INTEGER	PK	1,2..10,..n
login	VARCHAR(20)	UNIQUE	*admin, pavel007*
heslo	VARCHAR(20)		*adm1234, PaRa89*
jmeno	VARCHAR(20)		*Pavel, Rudolf, J.*
prijmeni	VARCHAR(30)		*Rakús, Jelínek, Novák*
email	VARCHAR(100)		*pavel.rakus@seznam.cz*
id_typ_uctu	INTEGER	FK	1,2..10,..n

V tabulce *ucty* jsou ukládány potřebné informace o uživatelích databázové aplikace pro podporu šachové hry. Kromě jedinečného ID je účet unikátně rozlišen také podle přihlašovacího jména (atribut *login*). Význam dalších textových atributů jako *jmeno*, *prijmeni* či *email* je na první pohled jasný z jejich názvu. Délka atributu *email* byla nastavena na hodnotu 100 znaků, aby byla ponechána dostatečná rezerva pro případnou velmi dlouhou e-

mailovou adresu. Samozřejmě by bylo možné ukládat o uživateli i další dodatečné informace, pro jednoduchost registrace jsou však ve formuláři vyžadovány pouze tyto.

Tabulka 3.5 - Tabulka "typ_uctu"

Tabulka „typ_uctu“			
Název atributu	Datový typ	Vlastnosti	Příklady hodnot
id_typ_uctu	INTEGER	PK	1,2..10,..n
nazev_uctu	VARCHAR(20)		*administrátor, uživatel*
prava	SMALLINT		1, 2, 3

Tabulka *typ_uctu* je určena pro uchovávání informací o obecných vlastnostech účtů a jejich právech. Primární klíč tabulky, atribut *id_typ_uctu*, je zároveň cizím klíčem a spojovacím prvkem s tabulkou *ucty*. Atribut *prava*, reprezentovaný datovým typem `SMALLINT`, je zde využíván pro číselné označení úrovně možností přístupu do aplikace. Nejvyšší (plná) práva jsou reprezentována hodnotou 1 (administrátor), naopak nejnižší práva pak hodnotou 3 (běžný uživatel). Samozřejmě je zde prostor pro případné rozšíření o další možné typy uživatelských nebo speciálních účtů.

Tabulka 3.6 - Tabulka "hraci"

Tabulka „hraci“			
Název atributu	Datový typ	Vlastnosti	Příklady hodnot
id_hrace	INTEGER	PK	1,2..10,..n
prijmeni	VARCHAR(30)		*Rakus, Novak, Vavrinek*
jmeno	VARCHAR(20)		*Pavel, Jan, Radomir*

V tabulce *hraci* jsou uchovávána pouze základní data o jednotlivých hráčích – konkrétně jméno a příjmení. Hodnoty obou těchto atributů jsou v podobě textového řetězce bez diakritiky, což vychází z předpisu formátu PGN.

Následující dvě tabulky jsou v podstatě shodné, přesto je potřeba, aby byly realizovány odděleně.

Tabulka 3.7 - Tabulka "hrac_bily"

Tabulka „hrac_bily“			
Název atributu	Datový typ	Vlastnosti	Příklady hodnot
id_hrac_bily	INTEGER	PK	1,2..10,..n
elo	VARCHAR(4)		*2130, 1854*
id_hrace	INTEGER		1,2..10,..n

Tabulka 3.8 - Tabulka "hrac_cerny"

Tabulka „hrac_cerny“			
Název atributu	Datový typ	Vlastnosti	Příklady hodnot
id_hrac_cerny	INTEGER	PK	1,2..10,..n
elo	VARCHAR(4)		*2130, 1854*
id_hrace	INTEGER		1,2..10,..n

Atributy *id_hrac_bily* resp. *id_hrac_cerny*, jsou jedinečné identifikátory každého hráče, který hrál v nějaké partii bílými, resp. černými figurami. V atributu *elo* je uchována hodnota jeho aktuálního FIDE elo ratingu. Položka *id_hrace* je cizím klíčem, propojujícím tabulku *hrac_bily* resp. *hrac_cerny* s tabulkou *hraci*. Atribut *id_hraci* stojí v relaci 1:N na straně N, což znamená, že tabulky *hrac_bily* a *hrac_cerny* mohou obsahovat více položek se stejným hráčem. U každého takového záznamu je pak rozdílná položka *elo*, která se v praxi může měnit v měsíčních nebo čtvrtletních intervalech.

Tabulka 3.9 - Tabulka "udalosti"

Tabulka „udalosti“			
Název atributu	Datový typ	Vlastnosti	Příklady hodnot
id_udalosti	INTEGER	PK	1,2..10,..n
nazev_udalosti	VARCHAR(100)		*2LF 2013/14 Opava-Brno*
misto	VARCHAR(100)		*Opava, Dolni Benesov*

Udalosti je tabulka s informacemi o názvu události (turnaj, zápas) a místě, kde byla konkrétní partie sehrána. Délka obou atributů (*nazev_udalosti, místo*) je z rezervních důvodů nastavena na hodnotu 100, což by mělo bezpečně pokrýt i případný delší název šachové události resp.

podrobné udání místa události (např. včetně zahrnutí názvu objektu, adresy, městské částí apod.).

Tabulka 3.10 - Tabulka "kod_zahajeni"

Tabulka „kod_zahajeni“			
Název atributu	Datový typ	Vlastnosti	Příklady hodnot
eco	VARCHAR(3)	PK	A00 – E99
nazev_zahajeni	VARCHAR(100)		*English, King's Indian defense*
typ_zahajeni	VARCHAR(100)		*otevrene, zavrene, polozavrene*

Celkem 500 položek je obsaženo v tabulce *kod_zahajeni*. Každý záznam tabulky představuje jednu z možných variant šachového zahájení. Definován je pomocí hodnoty tzv. ECO kódu šachového zahájení, který je složen ze znaku ‚A' až ‚E' a dvou číslic v rozmezí 0 – 9. Atribut *eco* je primárním klíčem tabulky, atribut *nazev_zahajeni* pak slouží k slovnímu popisu názvu daného zahájení a atributem *typ_zahajeni* je zahájení určeno jako otevřené, polozavřené nebo zavřené.

Tabulka 3.11 - Tabulka "partie"

Tabulka „partie“			
Název atributu	Datový typ	Vlastnosti	Příklady hodnot
id_partie	INTEGER	PK	1,2..10,..n
datum	VARCHAR(10)		*2008.12.1, 2014.4.14*
kolo	VARCHAR(5)		*1.1, 9.45, 11.11*
eco	VARCHAR(3)	FK	A00 – E99
vysledek	VARCHAR(10)		1-0, 1/2-1/2, 0-1, *
tahy	TEXT		1. e4 c5 2. Nf3 Nf6 ...
id_udalosti	INTEGER	FK	1,2..10,..n
id_uctu	INTEGER	FK	1,2..10,..n
id_hrac_bily	INTEGER	FK	1,2..10,..n
id_hrac_cerny	INTEGER	FK	1,2..10,..n

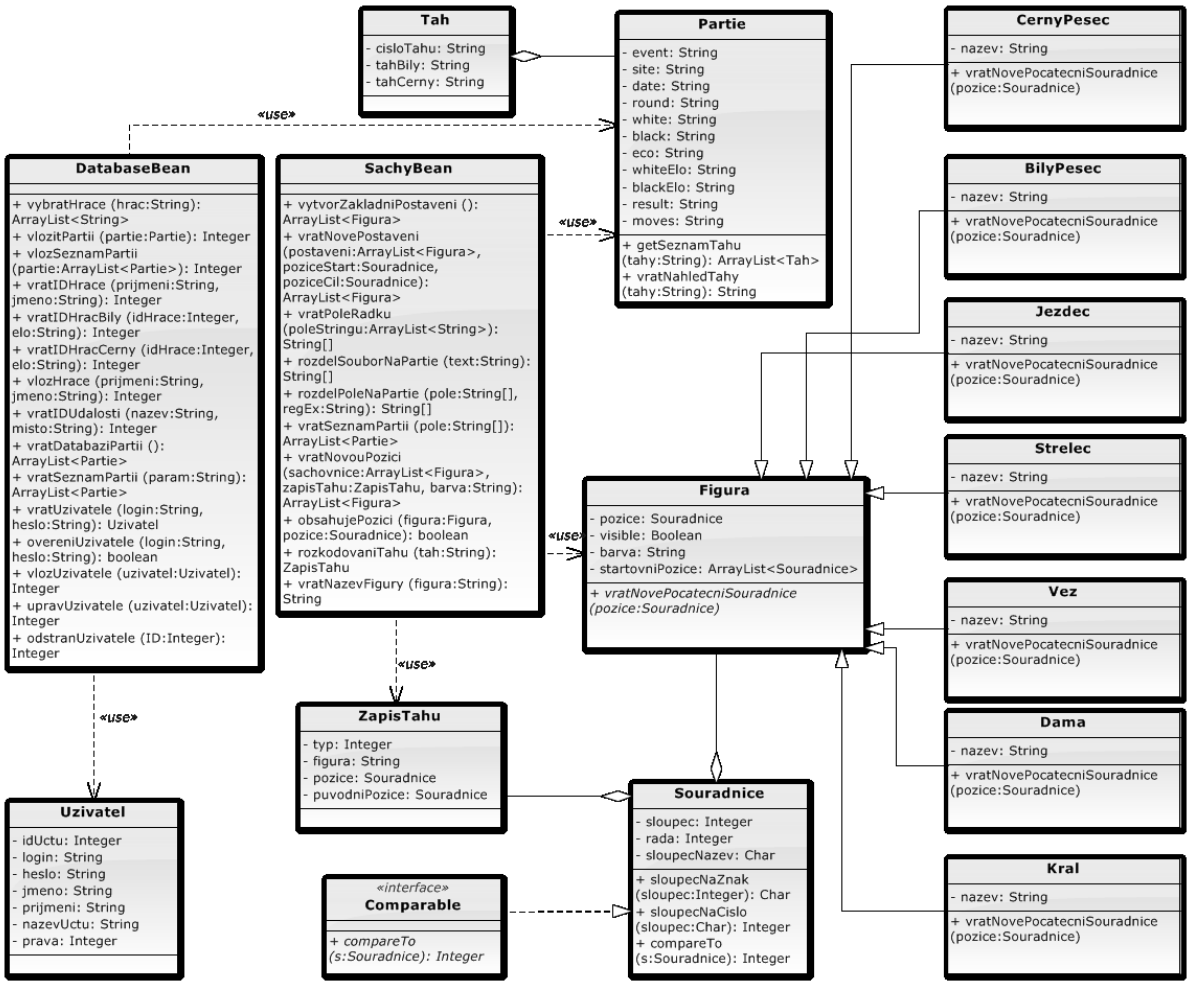
Tabulka *partie* je nejdůležitějším objektem ve struktuře relační databáze. Jedná se o propojovací tabulku, obsahující cizí klíče (Foreign Key, FK) a relace typu 1:N se všemi

ostatními hlavními tabulkami. Kromě primárního klíče s vlastností `IDENTITY(1,1)` – což je mimochodem vlastnost nastavená u všech tabulek, kde je primární klíč reprezentován hodnotou `INTEGER` – je v ní obsaženo pět cizích klíčů a čtyři vlastní atributy. Atribut datum je zde realizován jako `VARCHAR`. Důvodem pro tuto skutečnost je opět předpis formátu PGN, kterým je umožněno do položky *„Date“* vkládat kromě kompletního data také datum nekompletní nebo dokonce žádné. Pod atributem *kolo* lze nalézt označení aktuální kola turnaje nebo aktuálního kola utkání v soutěži družstev a označení konkrétní šachovnice (u soutěží družstev 1 – 8), které je odděleno tečkou. Atribut *vysledek* je rozhodující pro správné určení výsledku partie, kromě klasických hodnot „1-0“, „1/2-1/2“ a „0-1“ může nabývat také hodnoty „*“. Posledním atributem je položka *tahy*, ve které je uložen celý zápis partie. Zápis partie může být samozřejmě velice dlouhý, proto je zde použit datový typ `TEXT`. Pro uložení záznamu partie by samozřejmě postačovalo využití této tabulky, tento způsob realizace by ovšem odporoval základním principům relační databáze, neboť by docházelo k velké redundanci dat.

3.3 Realizace Java Enterprise Aplikace

K realizaci samotné Java Enterprise webové aplikace bylo využito vývojového prostředí NetBeans IDE 7.1 (s JDK v1.7) a serveru GlassFish verze 3.1.1 s doplněnými JSQLConnect drivery pro připojení k Microsoft SQL serveru.

3.3.1 Diagram tříd aplikace



Obrázek 3.3 - Diagram tříd databázové aplikace pro podporu šachové hry

Na obrázku 3.3 je znázorněn diagram tříd aplikace, ze kterého lze vyčíst základní parametry jednotlivých tříd a vazby mezi nimi. Základ je tvořen dvěma session enterprise beany (`SachyBean`, `DatabaseBean`). Komponenta `SachyBean` využívá ke své činnosti třídu `Figura` (a také všechny její podtřídy), dále třídy `ZapisTahu`, `Souradnice`, `Partie` a `Tah`. Komponenta `DatabaseBean`, která je určena pro práci s databází, využívá třídu `Uzivatel` a dále třídy `Partie` a `Tah`. Do této komponenty je také pomocí anotace `@Resource` namapován odkaz na databázový zdroj „*dbsachy*“ třídy `DataSource` z balíčku `javax.sql`, pomocí kterého jsou prováděny základní funkce databáze.

3.3.2 Popis a funkce vybraných tříd aplikace

Z výše uvedeného diagramu tříd lze třídy v aplikaci rozdělit podle jejich důležitosti (funkce) na dvě skupiny: *hlavní* a *pomocné*. Za hlavní třídy, kterými jsou vykonávány

metody, a které komunikují se servlety, řídícími celou aplikaci, lze považovat oba enterprise beany – `SachyBean` a `DatabaseBean`. Ostatní třídy jsou pomocné a jsou těmito beany využívány.

Tabulka 3.12 - Podrobný popis třídy DatabaseBean

`@Stateful` **DatabaseBean**	
Implementuje:	**DatabaseBeanLocal**

Importuje:	`java.sql.*;`
	`java.util.ArrayList;`
	`java.util.Arrays;`
	`javax.annotation.Resource;`
	`javax.ejb.Stateful;`
	`javax.sql.DataSource;`

Datové položky	
`DataSource dbsachy;`	Odkaz na databázový zdroj.
`Connection con;`	Připojení k databázi typu `Connection`.
`Statement stat;`	Objekt pro vykonávání statického SQL příkazu třídy `Statement`.
`PrepatedStatement prstat;`	Předkompilovaný SQL `Statement`, určený pro parametrizované dotazy.
`ResultSet rs;`	Tabulka dat, získaná provedením SQL dotazu.
Metody	
`ArrayList<String>` **`vybratHrace`**`(String hrac)`	Vrací seznam hráčů z tabulky `hraci` na základě zadaného parametru.
`int` **`vlozitPartii`**`(Partie partie)`	Vloží partii do tabulky `partie`. Vrací 1 v případě úspěšného provedení.
`int` **`vlozSeznamPartii`** `(ArrayList<Partie> partie)`	Vloží seznam partií do tabulky `partie`. Vrací počet řádků úspěšně vložených do databáze.
`int` **`vratIDHrace`** `(String prijmeni, String jmeno)`	Vrací ID hráče z tabulky `hraci`, u kterého je shoda s parametry *prijmeni* a *jmeno*.
`int` **`vratIDHracBily`** `(int idHrace,`	Vrací ID hráče z tabulky `hrac_bily`,

String elo)	hrajícího v konkrétní partii bílými figurami, podle zadaného *ELA* a *ID* ze seznamu hráčů.
int **vratIDHracCerny** (int idHrace, String elo)	Vrací *ID* hráče z tabulky hrac_cerny, hrajícího v konkrétní partii černými figurami, podle zadaného *ELA* a *ID* ze seznamu hráčů.
int **vlozHrace** (String prijmeni, String jmeno)	Vloží hráče do tabulky hraci. Vrací 1 v případě úspěšného provedení.
int **vratIDUdalosti** (String nazev, String místo)	Vrací ID události z tabulky udalosti podle zadaného názvu události a místa konání.
ArrayList<Partie> **vratDatabaziPartii** ()	Vrací seznam všech partií, uložených v databázi v tabulce partie.
ArrayList<Partie> **vratSeznamPartii** (String param)	Vrací seznam partií z tabulky partie, kde partie obsahuje zadaný parametr.
Uzivatel **vratUzivatele** (String login, String heslo)	Vrací novou instanci třídy Uzivatel z tabulky Ucty dle zadaného *loginu* a *hesla*.
boolean **overeniUzivatele** (String login, String heslo)	Vrací hodnotu true, jestliže je v tabulce ucty nalezena shoda *loginu* a *hesla* se zadanými parametry.
int **vlozUzivatele** (Uzivatel uživatel)	Vloží uživatele do tabulky ucty. Vrací 1 v případě úspěšného provedení.
int **upravUzivatele** (Uzivatel uživatel	Upraví údaje o uživatelském účtu v tabulce ucty.
int **odstranUzivatele** (int ID)	Vymaže z tabulky ucet záznam dle zadaného *ID*.

Tato výše popsaná komponenta je využívána pro práci s databází, sama využívá především tříd Partie a Uzivatel. Pokud by měla být zmíněna nějaká klíčová metoda této komponenty (třídy), pravděpodobně by se jednalo o metodu vratSeznamPartii(). Tato metoda je volána, pokud uživatel v servletu zadá textový parametr a potvrdí jej kliknutím na tlačítko „Vyhledat partie v databázi“. Návratovým typem metody je kolekce objektů třídy Partie, se kterou lze nadále pracovat (nejčastěji nahrát do seznamu pro výběr partie a následně přehrát zvolenou konkrétní partii).

Druhou, a co do funkčnosti a obsáhlosti ještě významnější komponentou, je enterprise bean s názvem `SachyBean`. V tabulce níže je opět tato třída konkrétně popsána.

Tabulka 3.13 - Podrobný popis třídy SachyBean

`@Stateful` **SachyBean**	
Implementuje:	**SachyBeanLocal**

Importuje:	`java.util.ArrayList;`
	`java.util.Arrays;`
	`javax.ejb.Stateful;`

Datové položky	
Tato třída nemá deklarovány žádné globální datové položky.	
Metody	
`ArrayList<Figura>` **`vytvorZakladniPostaveni`** `()`	Vrací kolekci figur s konkrétně nastavenými atributy (barva, pozice, přípustné tahy).
`ArrayList<Figura>` **`vratNovePostaveni`** `(ArrayList<Figura> postaveni, Souradnice poziceStart, Souradnice poziceCil)`	Vrací novou kolekci figur na základě dodaných parametrů. V podstatě se jedná o provedení tahu při zadání počátečních a koncových souřadnic.
`String[]` **`vratPoleRadku`** `(ArrayList<String> poleStringu)`	Vrací pole objektů typu `String`.
`String[]` **`rozdelSouborNaPartie`** `(String text)`	Rozdělí nahraný textový soubor na dočasné pole.
`String[]` **`rozdelPoleNaPartie`** `(String[] pole, String regEx)`	Rozdělí pole podle zadaného výrazu na konkrétní pole partií.
`ArrayList<Partie>` **`vratSeznamPartii`** `(String[] pole)`	Vrací kolekcí objektů typu `Partie` na základě dodaného pole partií.
`ArrayList<Figura>` **`vratNovouPozici`** `(ArrayList<Figura> sachovnice, ZapisTahu zapisTahu, String barva)`	Vrací novou kolekci objektů typu `Figura` s nastavenými parametry na základě dodané původní kolekce figur, zápisu tahu a barvy figury.
`boolean` **`obsahujePozici`** `(Figura figura, Souradnice pozice)`	Vrací hodnotu `true`, jestliže pole přípustných pozic figury obsahuje parametr

	pozice.
ZapisTah **rozkodovaniTahu** (String tah)	Vrací nový objekt typu ZapisTahu na základě textového řetězce se zápisem tahu.
String **vratNazevFigury** (String figura)	Vrací celý název figury na základě dodané znakové zkratky.

Hlavní úlohou této komponenty je ovládat nejdůležitější část aplikace - samotný přehrávač partií. Pro vytvoření šachovnice je určena metoda vytvorZakladniPostaveni(), ve které je vytvořeno všech 32 objektů různých tříd - 2x Kral, 2x Dama, 4x Vez (2x bílá a 2x černá), 4x Strelec (2x bílý a 2x černý), 4x Jezdec (2x bílý a 2x černý), 8x BilyPesec a 8x CernyPesec. U figur je jejich barva rozlišena pomocí datové položky třídy Figura s názvem barva typu String, u pěšců je barva rozlišena přímo v názvu třídy. Důvodem je metoda vratNovePocatecniSouradnice(), která je pro bílého a černého pěšce odlišná. Každá figura vrací stejné nové počáteční souřadnice bez ohledu na barvu (jinak řečeno bílá i černá figura táhne stejně), kdežto bílý pěšec se může pohybovat ve směru od 2. po 8. řadu, zatímco černý pěšec ve směru od 7. po 1. řadu. Všechny třídy těchto objektů jsou potomky třídy Figura, čehož lze využít při konstrukci kolekce figur. Metoda vytvorZakladniPostaveni() tedy vrací kolekci ArrayList<Figura>, do které jsou v metodě všechny konkrétní objekty přidány.

Další důležité metody jsou ty pro práci s textem. V servletu jsou znaky ze souboru ve formátu PGN načteny do objektu typu String, který musí být dále zpracován. Pro tento účel je vytvořena metoda rozdelSouborNaPartie(), ve které je soubor rozdělen na dočasné pole partií. Pro rozdělení tohoto dočasného pole na konkrétní partie je pak použita metoda rozdelPoleNaPartie(), ve které je podle určitého řetězce („1/2-1/2“ nebo „0-1“) pole rozděleno. Na základě tohoto pole, které je parametrem metody vratSeznamPartii() je vytvořena kolekce objektů typu Partie, a ta je touto metodou navrácena a je s ní dále pracováno.

Pro získání konkrétního tahu je určena metoda rozkodovaniTahu(). Jejím parametrem je konkrétní jednotlivý tah, který může mít až překvapivě mnoho variant. Návratovým typem této metody je instance třídy ZapisTahu, jejíž definici je vhodné na tomto místě uvést ještě před samotným kódem metody rozkodovaniTahu().

Tabulka 3.14 - Popis třídy ZapisTahu

ZapisTahu	
Implementuje:	**Serializable**
Importuje:	`java.io.Serializable;`
Datové položky	
`int typ;`	Číselné označené typu tahu (viz tabulka 3.15).
`String figura;`	Znakové označení figury, která tah provádí.
`Souradnice pozice;`	Souřadnice koncové pozice typu `Souradnice`.
`Souradnice puvodniPozice;`	Souřadnice původní pozice typu `Souradnice`.
`String novaFigura;`	Znakové označení nově vzniklé figury při proměně pěšce.
`char puvodniSloupec;`	Znakové označení původního sloupce, odkud figura táhla.
`int puvodniRada;`	Číselné označení původní řady, odkud figura táhla.
Metody	
Tato třída nemá žádné metody.	
*(Pozn.: V seznamu metod nejsou uvedeny **getry** a **setry**.)*	
Konstruktory	
`ZapisTahu``()`	
`ZapisTahu``(int typ, String figura, Souradnice pozice, String novaFigura)`	
`ZapisTahu``(int typ, String figura, Souradnice pozice, char puvodniSloupec)`	
`ZapisTahu``(int typ, String figura, Souradnice pozice, int puvodniRada)`	

Velice důležitou datovou položkou je číselná hodnota `typ`. Tato proměnná určuje, o jaký konkrétní druh tahu se jedná, a je podle ní větvena metoda `vratNovouPozici()`, v níž je

v podstatě na základě dodaných parametrů proveden určitý tah. Logika této metody je podrobněji popsána v kapitole 3.3.4.

Tabulka 3.15 - Možné typy tahů v šachové partii

Typy tahů		
Číselné označení typu	Název typu	Příklad
1	standardní tah	e4, Jc6, Kg8
2	rozšířený tah (shoda cílového pole)	Jce4, Vae1
3	braní figurou (obsahuje „x“)	Dxc5
4	nová figura (obsahuje „=“)	e8=D, a8=J
5	rozšířené braní (shoda cílového pole)	Jexd5, Vaxc1
6	braní a nová figura (obsahuje „x“ i „=“)	axb8=Q, hxg8=N
7	malá rošáda	O-O
8	velká rošáda	O-O-O
9	braní pěšcem	bxc6, axb4

3.3.3 Kód a logika vybrané metody komponenty SachyBean

Pro ukázku kódu byla zvolena metoda komponenty `SachyBean` s názvem `rozkodovaniTahu()`. Metoda vrací obecnou výjimku, pokud je špatně zadán její parametr `tah`. Kód metody může být uveden v následující podobě:

```
public ZapisTahu rozkodovaniTahu(String tah) throws Exception {
        int typ = 0;
        String figura = "";
        Souradnice pozice = new Souradnice('a', 1);
        if (tah.contains("+")) {
            tah = tah.substring(0, tah.indexOf("+"));
        }
        switch (tah.length()) {
            case 0:
            case 1: {
                throw new Exception("Zadaný tah není korektní!");
            }
            case 2: {
                typ = 1;
                figura = "P";
                pozice.setSloupecNazev(tah.charAt(0));
                pozice.setRada(Integer.parseInt(tah.substring(1, 2)));
            }
            break;
            case 3: {
                if (tah.contains("-")) {
                    return new ZapisTahu(7, "K", pozice);
                }
                typ = 1;
                figura = tah.substring(0, 1);
```

```
    pozice.setSloupecNazev(tah.charAt(1));
    pozice.setRada(Integer.parseInt(tah.substring(2, 3)));
}
break;
case 4: {
    if (tah.contains("x")) {
        if (Character.isLowerCase(tah.charAt(0))) {
            typ = 9;
            figura = "P";
            pozice.setSloupecNazev(tah.charAt(2));
            pozice.setRada(Integer.parseInt(tah.substring(3, 4)));
            char puvodniSloupec = tah.charAt(0);
            return new ZapisTahu(typ, figura, pozice, puvodniSloupec);
        } else {
            figura = tah.substring(0, 1);
            typ = 3;
            pozice.setSloupecNazev(tah.charAt(2));
            pozice.setRada(Integer.parseInt(tah.substring(3, 4)));
        }
    } else if (tah.contains("=")) {
        typ = 4;
        figura = "P";
        pozice.setSloupecNazev(tah.charAt(0));
        pozice.setRada(Integer.parseInt(tah.substring(1, 2)));
        String novaFigura = tah.substring(3, 4);
        return new ZapisTahu(typ, figura, pozice, novaFigura);
    } else {
        typ = 2;
        figura = tah.substring(0, 1);
        pozice.setSloupecNazev(tah.charAt(2));
        pozice.setRada(Integer.parseInt(tah.substring(3, 4)));
        if (Character.isDigit(tah.charAt(1))) {
            int puvodniRada = Integer.parseInt(tah.substring(1, 2));
            return new ZapisTahu(typ, figura, pozice, puvodniRada);
        } else {
            char puvodniSloupec = tah.charAt(1);
            return new ZapisTahu(typ, figura, pozice, puvodniSloupec);
        }
    }
}
break;
case 5: {
    if (tah.contains("-")) {
        return new ZapisTahu(8, "K", pozice);
    } else if (tah.contains("x")) {
        typ = 5;
        figura = tah.substring(0, 1);
        pozice.setSloupecNazev(tah.charAt(3));
        pozice.setRada(Integer.parseInt(tah.substring(4, 5)));
        if (Character.isDigit(tah.charAt(1))) {
            int puvodniRada = Integer.parseInt(tah.substring(1, 2));
            return new ZapisTahu(typ, figura, pozice, puvodniRada);
        } else {
            char puvodniSloupec = tah.charAt(1);
            return new ZapisTahu(typ, figura, pozice, puvodniSloupec);
        }
    } else {
        throw new Exception("Zadaný tah není korektní!");
    }
}
case 6: {
    if (tah.contains("=")) {
        typ = 6;
        figura = "P";
        pozice.setSloupecNazev(tah.charAt(2));
        pozice.setRada(Integer.parseInt(tah.substring(3, 4)));
        String novaFigura = tah.substring(5, 6);
        return new ZapisTahu(typ, figura, pozice, novaFigura);
```

```
                } else {
                    throw new Exception("Zadaný tah není korektní!");
                }
            }
        }
        return new ZapisTahu(typ, figura, pozice);
    }
```

Na samotném začátku metody je zajištěna inicializace datových položek `typ` a `figura` a vytvořena nová instance třídy `Souradnice` s názvem `pozice`. V dalším kroku je pak testován výskyt znaku „+“, indikující označení šachu, v řetězci. Vzhledem k tomu, že tento příznak nemá vliv na tah figury (je uvažován pouze korektní zápis partie ve formátu PGN), je ignorován, a původní parametr tah typu `String` je nahrazen novým objektem `tah` typu `String`, zkráceným právě o tento znak (dle normy je vždy uveden na konci zápisu tahu).

Dále je metoda větvena podle délky tahu. Pokud je délka rovna 0 nebo 1, metoda vyhodí novou výjimku s hlášením „Zadaný tah není korektní“. Tento případ ale v podstatě nemůže nastat kvůli předpisu formátu PGN.

Pokud je délka rovna 2, jedná se zcela jistě o tah pěšce (typ 1). Je provedeno nastavení sloupce v proměnné pozice na název získaný jako znak na 1. pozici řetězce tah a nastavení řady v proměnné pozice na číselnou hodnotu přetypovaného znaku na 2. pozici řetězce tah (je uvažován pouze korektní zápis tahu v souboru PGN). Tento postup je analogicky prováděn u dalších větví v této metodě.

Pokud je délka tahu rovna hodnotě 3, je ověřováno, zda řetězec tah obsahuje znak pomlčky. V takovém případě se jedná o malou rošádu (typ tahu 7) a může být vrácena nová proměnná typu `ZapisTahu`. V případě, že řetězec tah pomlčku neobsahuje, jedná se o standardní tah figury (typ tahu 1), kde na první pozici řetězce `tah` je znakové označení figury, na 2. pozici znakové označení sloupce a na 3. pozici číselné označení řady.

V případě, že je délka řetězce `tah` rovna hodnotě 4, je ověřováno, zda uvedený řetězec obsahuje znak „x“. Platí - li to, je testováno, zda je znak na první pozici malé písmeno. Pokud ano, jedná se o braní pěšcem, kdy na 1. pozici v řetězci je označení původního sloupce, na 2. pozici znaménko indikující braní, na 3. pozici označení sloupce a na 4. pozici označení řady. Pro figuru je toto stejné s drobným rozdílem, totiž že na 1. pozici je znakové označení figury místo původního sloupce. V případě, že je délka řetězce rovna 4, a zároveň řetězec obsahuje znaménko „=“, indikující proměnu pěšce ve figuru na 8. (1.) řadě, jedná se o typ tahu 4. Opět jsou nastaveny hodnoty v proměnné pozice, na 1. pozici řetězce je označení původního

sloupce, na 2. pozici označení řady, na 3. pozici znaménko „=“ a na 4. pozici znakové označení nové figury. Je - li hodnota délky řetězce 4 a neobsahuje - li ani znaménko „x“ ani „=“, jedná se o rozšířený tah, kde na cílové pole mohou táhnout zároveň 2 figury, a musí být rozlišeno, která z figur to bude. Zde je na 1. pozici znakové označení figury, na 2. pozici znakové označení původního sloupce nebo číselné označení původní řady (často u věží, pokud obě stojí na stejném sloupci, ale i u jezdců nebo příp. více dam), což musí být také pomocí podmínky testováno, na 3. pozici znakové označení sloupce a na 4. pozici číselné označení řady koncové pozice. Na konci všech větvení je vždy navrácen nový objekt typu `ZapisTahu` konstruovaný na základě odpovídajícího konstruktoru.

Pokud je délka řetězce `tah` rovna 5, je testováno, zda obsahuje znaménko pomlčky. V případě, že ano, jedná se zcela jistě o velkou rošádu (typ tahu 8) a je okamžitě navrácen nový objekt typu `ZapisTahu`. V případě, že řetězec obsahuje znak „x“, jedná se o rozšířené braní, kde musí být stejně jako u rozšířeného tahu uvedena doplňující informace o původním sloupci nebo původní řadě. Opět je testováno, zda je znak na 2. pozici v řetězci `tah` číselný či není. Na 1. pozici je pak uvedeno znakové označení figury (jezdec, věž nebo dáma), na 2. pozici znak původního sloupce nebo hodnota původní řady, na 3. pozici znak „x“, na 4. pozici znak sloupce a na 5. pozici hodnota řady cílové pozice.

Je – li délka tahu rovna hodnotě 6, jedná se zcela jistě o braní pěšcem zároveň s proměnnou na figuru na 8. či 1. řadě (typ tahu 6), kde na 1. pozici řetězce je znak původního sloupce pěšce, na 2. pozici znaménko „x“, na 3. pozici znak sloupce cílové pozice, na 4. pozici hodnota řady cílové pozice, na 5. pozici znaménko „=“, indikující vznik nové figury a na 6. pozici znakové označení nové figury (nejčastěji „D“ resp. „Q“).

Vzhledem k předpisu formátu PGN jsou očekávány korektní řetězce tahů daných typů. Tento předpoklad je navíc podložen skutečností, že drtivá většina souborů ve formátu PGN je generována automaticky softwarem při snímání tahů z digitální šachovnice, což zabezpečuje korektnost zápisu, nebo tvořena uživatelem v některém z pokročilých šachových softwarů, kde je také zaručena 100% korektnost daného zápisu.

3.3.4 Logika metody provedení tahu komponenty SachyBean

Hlavní metodou celé komponenty `SachyBean` je metoda `vratNovouPozici()`, ve které je na základě dodaných parametrů nastavena nová pozice pro zobrazení na příslušné JSP stránce. Vstupními parametry metody jsou následující položky:

- `ArrayList<Figura> šachovnice` – kolekce 32 objektů s nastavenými parametry,
- `ZapisTahu zapisTahu` – objekt typu `ZapisTahu`, vrácený metodou `rozkodovaniTahu()`,
- `String barva` – barva hráče, jehož tah má být v metodě proveden.

V samotné metodě je na začátku po vytvoření nezbytných objektů opět větvení, tentokrát podle typu tahu, získaného z objektu `zapisTahu`. Logický postup metody je následující:

1) Vytvoření nového objektu s názvem `nazevFigura` typu `String`, získaného z kompletního názvu figury pomocí metody `vratNazevFigury()`.
2) Vytvoření nového objektu `pozice` typu `Souradnice` (např. 1,1 nebo 2,3 až po 8,8) na základě hodnot z parametru `zapisTahu`.
3) Větvení podle typu tahu, získaného z parametru `zapisTahu`.
    a. Průchod kolekcí `sachovnice` typu `Figura`.
    b. Testování, zda konkrétní třída objektu typu `Figura` obsahuje datovou položku `nazevFigura`.
    c. Testování shody nastavených parametrů konkrétní figury z kolekce objektů typu `Figura` s parametry, získanými z objektu `zapisTahu` a pomocí metody `obsahujePozici()` s parametrem pozice.
    d. Nastavení pozice konkrétní figury na hodnoty proměnné `pozice`.
    e. Nastavení nových počátečních souřadnic figury pomocí metody `vratNovePocatecniSouradnice()` s parametrem aktuálně nastavené pozice.
    f. Provedení případných dalších operací, dle typu tahu.
        i. Nastavení viditelnosti objektu se souřadnicemi shodnými jako u parametru `zapisTahu` z kolekce `sachovnice` na `false`, pokud je typ tahu braní.

ii. Vytvoření nového objektu jedné z podtříd třídy `Figura` s parametry získanými z objektu `zapisTahu` a její přidání do kolekce `sachovnice`, pokud je typ tahu proměna pěšce (zároveň nastavení viditelnosti pěšce na `false`).

g. Vrácení kolekce `sachovnice` s nově nastavenými parametry jednotlivých objektů příp. nově přidanými objekty.

4) Vrácení kolekce `sachovnice` s nově nastavenými parametry jednotlivých objektů, pokud tak nebylo učiněno již v bodě 3) části g.

Všechny příkazy metody jsou prováděny v bloku `try/catch` pro případ zachycení neočekávané výjimky (metoda vyhazuje obecnou výjimku typu `Exception`). Testováním shody nastavených parametrů v bodě 3) části c. je myšleno ověření, zda se shoduje barva figury s barvou předanou parametrem metody, dále případně původní sloupec nebo původní řada s hodnotami z parametru `zapisTahu`, zda je viditelnost (atribut `visible`) nastavena na hodnotu `true` a dále také již uvedené testy.

3.3.5 Řízení chodu aplikace

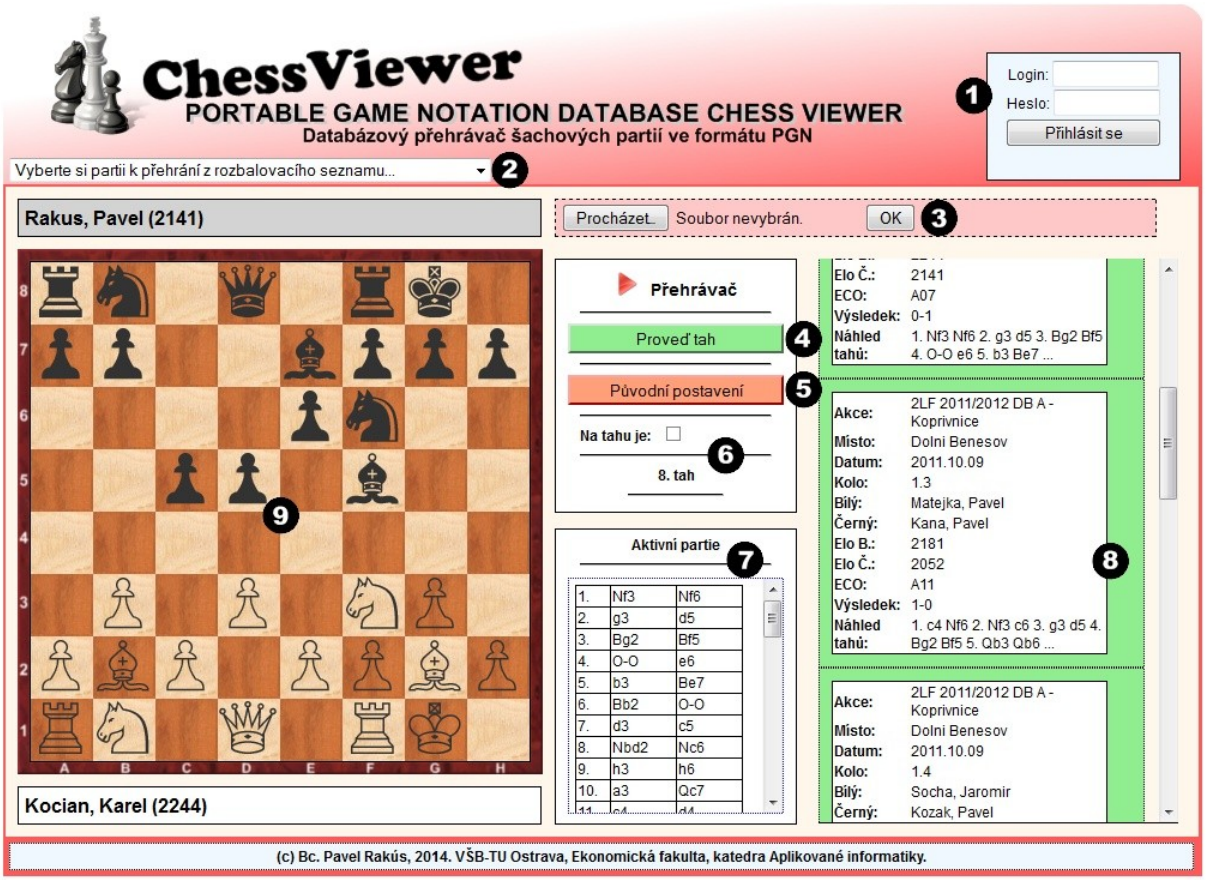
Řízení aplikace je prováděno za pomocí dvou servletů, `ServletSachy` a `DatabaseServlet`. Servletem `ServletSachy` je ke své činnosti využívána komponenta `SachyBean` a její metody, a slouží k řízení šachového přehrávače, nahrávání partií ze souboru a k dalším s tímto spojeným činnostem. Servletem `DatabaseServlet` je pak používána komponenta `DatabaseBean` a její metody, a tento servlet slouží pro práci s databází, tzn. výběr partií, vkládání uživatelů, úprava uživatelů atd. Při volání servletu z JSP stránky je vždy odeslán parametr akce, který je využíván k počátečnímu větvení. V každé větvi jsou pak provedeny požadované metody a nastaveny atributy, které jsou dále odeslány na cílovou JSP stránku pomocí přesměrování. V častých případech je pro uchování atributů využívána proměnná `session` typu `HttpSession` z balíčku `javax.servlet.http`.

Po spuštění aplikace (v reálném prostředí by to znamenalo zadání konkrétní adresy do webového prohlížeče) je automaticky načtena JSP stránka `index.jsp`. V těle této JSP stránky je pomocí značky ze sady `Core` knihovny JSTL okamžitě provedeno přesměrování na servlet `ServletSachy` s parametrem `akce` nastaveným na hodnotu 2. Tímto dojde k přesměrování na JSP stránku `prehravac.jsp`, která může být považována za vstupní

bránu aplikace. Uživatel má nyní několik možností, jak postupovat dále. Může se přihlásit, čímž dojde k volání servletu `DatabaseServlet` a přesměrování na stránku `admin.jsp`, nebo načíst partie ze souboru a tyto následně přehrát. Při odeslání každého požadavku je vždy zavolán příslušný servlet, požadavek je dle zadaných parametrů (proměnná `akce`) ošetřen a jsou nastaveny atributy, které jsou využity na cílové JSP stránce, zobrazené po přesměrování servletem ve webovém prohlížeči uživatele. Vzhled grafické prostředí aplikace je znázorněn v kapitole 3.3.6.

3.3.6 Základní funkce aplikace

Grafické uživatelské prostředí je vytvořeno za použití CSS stylů. Popis tvorby grafického rozhraní pro ovládání aplikace není předmětem této práce, nicméně měly by být popsány alespoň základní prvky a jejich funkce, vzhled grafické podoby stránky `prehravac.jsp` je zobrazen na obrázku 3.4.



Obrázek 3.4 - Grafické uživatelské rozhraní aplikace

Stránka `prehravac.jsp` je sestavena z těchto součástí:

(1) - formulář pro zadání přihlašovacího jména a hesla pro ověření uživatele,

(2) - rozbalovací seznam s nahranými partiemi (při změně je stránka odeslána),

(3) - formulář pro výběr souboru, jenž má být rozkódován a nahrán do seznamu partií,

(4) - akční tlačítko, pomocí kterého je proveden následující tah,

(5) - akční tlačítko, pomocí kterého je vytvořeno nové postavení (reset partie),

(6) - informace o aktuálním tahu a hráči, který je na tahu,

(7) - zápis aktivní partie (vybrané z rozbalovacího seznamu),

(8) - seznam partií, nahraných z vybraného souboru ve formátu PGN,

(9) - šachovnice v podobě tabulky s 8 řádky a 8 sloupci.

Postup při přehrání partie je velmi jednoduchý. V prvním kroku uživatel vybere ze svého počítače soubor ve formátu PGN a prostřednictvím formuláře (3) jej potvrdí. Aplikace provede potřebné akce a do seznamu partií (8) je nahrána kolekce objektů typů `Partie`, které jsou na JSP stránce pomocí JSTL značky `<c:forEach>` vypsány. Zároveň dojde k naplnění rozbalovacího seznamu (2) a změně jeho úvodního textu na „Vyberte si partii k přehrání z rozbalovacího seznamu...“. Uživatel seznam otevře a po výběru konkrétní partie se zobrazí její zápis v prostoru aktivní partie (7), je zpřístupněno tlačítko „Proveď tah“ (4) a jsou nahrána jména obou soupeřů. Pokud uživatel v této relaci již přehrával nějakou partii, jsou do výchozího stavu uvedeny informace o aktuální partii (6). Při každém stisknutí tlačítka „Proveď tah“ (4) je provedena série operací, jejímž výsledkem je vizuální změna pozice v tabulce šachovnice (9). Pokud je dosaženo posledního tahu v partii, je tlačítko „Proveď tah“ označeno jako neaktivní (`disabled`). Uživatel pak má možnost zvolit jinou partii ze seznamu aktivních partií (2), nebo tlačítkem „Původní postavení“ (5) opět přehrát aktivní partii znovu od začátku. Po přihlášení (1) může také využít funkcí databáze.

JSP stránka `prehravac.jsp` je hlavním stavebním prvkem grafického zobrazení aplikace. Využívá knihovnu JSTL, jejíž použití indikuje direktiva `@taglib`.

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

Záhlaví a zápatí je na všech stránkách v aplikaci shodné. V zápatí je uveden vždy pouze krátký text o autorovi, v záhlaví je zobrazen přihlašovací formulář, případně rozbalovací seznam s partiemi. Veškeré stránky JSP jsou uspořádány do následujícího rozložení:

```
    <div id="wrapper">
          <div id="header">
               hlavička stránky
          </div>
          <div id="content">
               vlastní obsah v těle stránky
          </div>
          <div id="footer">
               zápatí stránky
          </div>
    </div>
```

V těle stránky je využíváno značek knihovny JSTL k výpisu parametrů, získaných ze servletu nebo případně z proměnné `session` typu `HttpSession`, pravděpodobně nejzajímavější je vytvoření samotné šachovnice v tabulce.

```
<div class="sachy">
  <table style="border-collapse: collapse; border: 1px solid black;">
    <c:forEach begin="1" end="8" var="rada">
       <tr><c:forEach begin="1" end="8" var="sloupec">
            <td class="barva${(9 - rada + sloupec) % 2}">
              <c:forEach var="figura" items="${sachovnice}">
                <c:if test="${figura.pozice.sloupec == sloupec &&
                  figura.pozice.rada == (9 - rada) && figura.visible}">
                    <img src="figury/${figura.nazev}.png"alt="Figura">
                </c:if>
              </c:forEach>
            </td>
        </c:forEach></tr>
    </c:forEach>
  </table>
</div>
```

Ve dvou cyklech je testováno, zda aktuální souřadnice v cyklu odpovídají datové položce `pozice` z aktuálního objektu jedné z podtříd třídy `Figura`, a zda je datová položka `visible` tohoto objektu nastaven na `true`. Pokud platí obě uvedené podmínky, je v dané buňce tabulky zobrazen PNG obrázek s názvem získaným z datové položky `nazev` aktuálního objektu. Tímto způsobem je pak vytvořena kompletní tabulka. Po provedení tahu je servletem poskytnuta nová kolekce se změněnými parametry, která je opět takto vykreslena.

4 Využití aplikace v praxi a srovnání s ostatními produkty

V této kapitole je prostor věnován praktickému využití vytvořené aplikace a porovnání s ostatním softwarem se šachovou tématikou.

4.1 Praktické využití aplikace

Vzhledem k povaze aplikace se jako nejpravděpodobnější praktické využití nabízí implementace do již existujících webových stránek v podobě samostatného modulu. Pokud by byly do aplikace doplněny další stránky, mohla by být využita i jako samostatný web. Toto by ovšem vyžadovalo mírné doplnění databázové struktury a samozřejmě také úpravu grafické podoby s doplněním nových odkazů atd.

Jako nejpravděpodobnější uživatelé aplikace se jeví šachoví nadšenci, kteří chtějí využívat výhod rozsáhlé databáze partií, a zároveň nechtějí investovat prostředky do drahého vyspělého šachového softwaru. Vzhledem k zaměření databáze partií na Moravskoslezský kraj a jeho hráče budou uživatelé pravděpodobně příslušníci některého z šachových klubů právě v tomto kraji.

4.2 Srovnání s ostatními produkty

Softwaru, jenž je svou tématikou zaměřen na šachy, existuje samozřejmě spousta. Profesionální databáze od největšího producenta šachového softwaru, německé firmy ChessBase, jsou mimořádně kvalitní nástroj pro šachisty všech úrovní. Jako příklad dalšího produktu, zaměřeného na přehrávání šachových partií, může být uveden program ChessTheatre.

4.2.1 Databáze ChessBase

Databáze ChessBase je velmi užitečný software pro každého šachistu. V nejnovější verzi MegaBase 2014 je obsaženo více než 5.7 milionů partií z období let 1560 až 2013, z čehož je 67.500 doplněno komentáři špičkových hráčů. K těmto databázím je možné zakoupit i celou řadu různých šachových motorů. Šachový motor je velice sofistikovaný software, využívaný pro vlastní analýzu různých šachových pozic a partií. K nejlepším šachovým motorům v roce 2014 lze zařadit programy Deep Fritz 14, Deep Junior 13.8, Deep Rybka 4 a Houdini 4 Pro. [14]

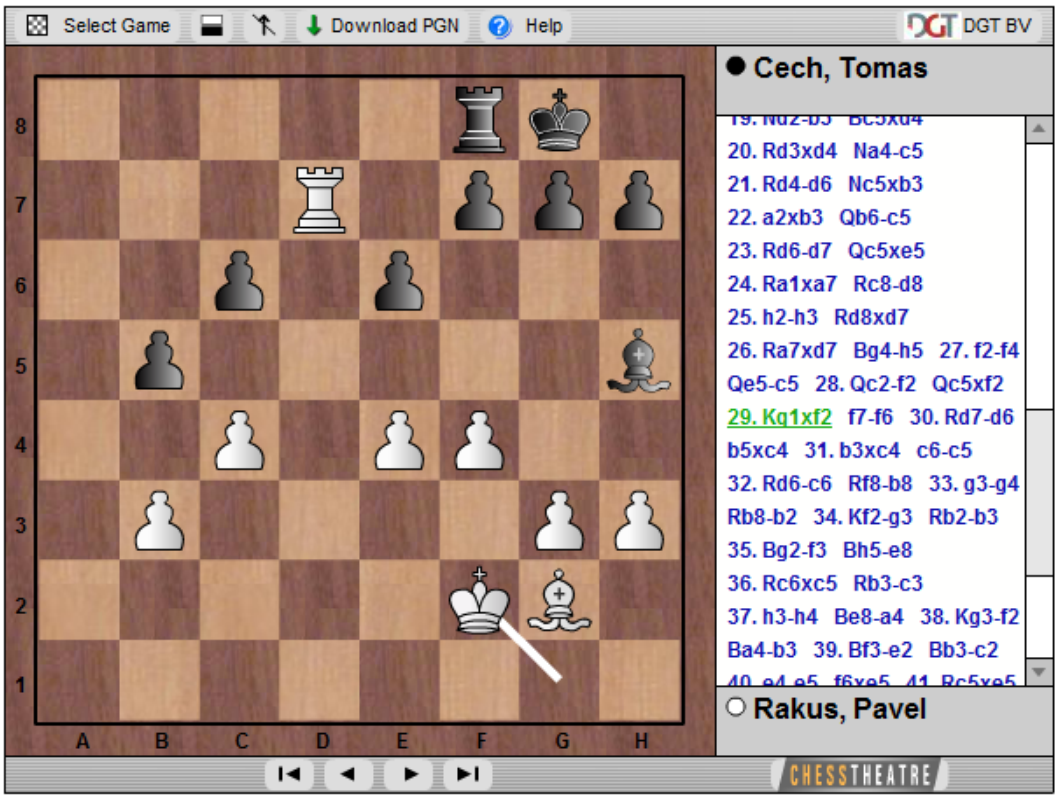
Výhody ChessBase: velice rozsáhlá databáze partií předních světových hráčů v období let 1560 až 2013, velký počet komentovaných partií pro lepší pochopení daných souvislostí a uvažování hráčů (komentáře především v anglickém a německém jazyce), rozsáhlá knihovna zahájení, silné vyhledávací nástroje, podpora dalších formátů (*.cbh, *.cba atd.), výborné uživatelské prostředí.

Nevýhody ChessBase: vysoká cena (160€ za databázi, motory v rozmezí 70 až 100€), neobsahuje partie hráčů na úrovni lokálních soutěží.

Srovnání s aplikací: Databáze ChessBase je samozřejmě na daleko vyšší úrovni, čemuž odpovídá i její vysoká cena. Aplikace je oproti tomu zaměřena hlavně na databázi partií uživatelů lokální úrovně, která naopak v MegaBase 2014 vůbec obsažena není. Aplikace je navíc přístupná z webového prohlížeče, tudíž není vázána na konkrétní platformu nebo hardware, což může být do jisté míry výhodou.

4.2.2 Šachový přehrávač ChessTheatre

Populární šachový přehrávač ChessTheatre je software, vyvíjený společností DGT Projects, která je mimo jiné výrobcem elektronických šachovnic, využívaný v předních



Obrázek 4.1 - Šachový přehrávač ChessTheatre společnosti DGT Projects

českých i světových soutěžích pro automatizovaný zápis a přenos šachových partií. Software byl vytvořen jako jednoduchý přehrávač právě pro záznamy šachových partií pořízených na hardwaru společnosti DGT Projects. Je vytvořen v technologii Flash a využíván jako doplňující modul ve webových stránkách, k dispozici je zdarma.

Výhody programu ChessTheatre: jednoduché a rychlé přehrávání partie, příjemný uživatelský vzhled, možnost stažení zdrojového PGN souboru, k dispozici zdarma.

Nevýhody programu ChessTheatre: potřeba příslušného Flash přehrávače, nutnost nahrávat zdrojové soubory do adresáře na webu, bez jakékoliv databáze (pracuje pouze s jednotlivými soubory), bez možnosti vyhledávání a třídění partií.

Srovnání s aplikací: Šachový přehrávač ChessTheatre je určen především jako jednoduchý nástroj pro přehrávání jednotlivého souboru PGN. Není jím využívána databáze, nepodporuje vyhledávání ani žádné řazení partií nahraných ze souboru. Grafické prostředí a možnosti přehrávání jsou na vyšší úrovni než u aplikace, informační a funkční hodnota ale kvůli neexistenci napojení na rozsáhlejší databázi partií chybí.

5 Závěr

Tvorba databázové aplikace pro podporu šachové hry probíhala v několika předem naplánovaných a na sebe navazujících krocích, vedoucích až ke splnění zadaného hlavního cíle.

Nejdříve bylo potřeba zjistit, jaká skupina uživatelů by mohla mít o využití takové aplikace zájem a jaké jsou jejich případné požadavky a náměty na funkcionalitu nově vznikající aplikace. V návrhu systému pak byly modelovány jednotlivé případy užití s hlavními aktéry, kteří v těchto případech vystupují.

V následujícím kroku muselo být rozhodnuto, v jakém konkrétním prostředí bude databázová aplikace pro podporu šachové hry implementována a jaké bude využívat technologie. Bylo zvoleno, že pro vývoj bude využito vývojového prostředí NetBeans IDE, a že celá aplikace bude pojata jako Java Enterprise Edition aplikace, využívající technologie Enterprise Java Beans.

Po zvolení technologie pro samotnou aplikaci bylo nutné zvolit také databázové prostředí, které bude aplikace využívat. Zde padla volba na Microsoft SQL Server technologii.

Po skončení fáze plánování byla zahájena samotná implementace aplikace v programovacím jazyce Java a realizace navrhnuté struktury databáze v prostředí Microsoft SQL Serveru. V současné době lze aplikaci nalézt ve stavu testování, kdy mohou být objeveny technické chyby, a před uvedením do provozu patřičně ošetřeny. Zároveň je možné ještě doplnit funkcionalitu aplikace o uživateli požadované funkce a rozšíření.

Jako pravděpodobné využití aplikace se jeví zakomponování do již existujících stránek šachového klubu, kde bude sloužit jako modul pro přehrávání partií pro registrované uživatele. Do budoucna je možné aplikaci rozšířit o nové funkce, případně vylepšit vzhled dle přání a požadavků uživatelů.

Seznam použité literatury

Knižní zdroje

[1] GRAND, Mark. *JAVA: Referenční příručka jazyka.* Brno: Computer Press, 1998. ISBN 80-7226-071-5.

[2] HEROUT, Pavel. *Java - grafické uživatelské prostředí a čeština.* 2. vyd. Kopp: České Budějovice, 2012. ISBN 978-80-7232-328-9.

[3] HEROUT, Pavel. *Učebnice jazyka Java.* Kopp: České Budějovice, 2004. ISBN 80-7232-115-3.

[4] JENDROCK, Eric, R. CERVERA-NAVARRO, I. EVANS, D. GOLLAPUDI, K. HAASE, W. MARKITO and CH. SRIVATHSA. *The Java EE 6 tutorial: advanced topics*. 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2010. ISBN 01-370-8186-3.

[5] LACKO, Ľuboslav. *Mistrovství v Microsoft SQL Server 2012.* Brno: Computer Press, 2013. ISBN 978-80-251-3773-4.

[6] LACKO, Ľuboslav. *SQL: kapesní přehled*. Brno: CP Books, 2005. ISBN 80-251-0788-4.

[7] LIANG, Y. Daniel. *Introduction to Java programming*. 8th ed. Upper Saddle River, NJ: Pearson Higher Education, 2011. ISBN 978-013-2130-790.

[8] SARANG, Poornachandra. *Java programming.* New York: McGraw-Hill, 2012. ISBN 978-0-07-163360-4.

[9] SCHILDT, Herbert. *Java 7: výukový kurz*. Brno: Computer Press, 2012. ISBN 978-80-251-3748-2.

Elektronické dokumenty

[10] ČÁPKA, David. *Úvod do Java Enterprise Edition.* DevBook.cz [online]. 2014, [cit. 2014-04-15]. Dostupné z: http://www.devbook.cz/java-enterprise-edition-uvod-do-jee-j2ee

[11] EDWARS, Steven. *Standard: Portable Game Notation Specification and Implementation Guide* [online]. 2000 [cit. 2014-04-18]. Dostupné z: http://www.saremba.de/chessgml/standards/pgn/pgn-complete.htm

[12] HORÁČEK, Petr. *Java na Webu III. – Servlety.* Linuxsoft.cz [online]. 7. 5. 2013 [cit. 2014-04-15]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=1974

[13] HORÁČEK, Petr. *Java na Webu IV. – JSP.* Linuxsoft.cz [online]. 16. 6. 2013 [cit. 2014-04-15]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=1978

[14] CHESSBASE. *ChessBase Shop: Mega Database 2014* [online]. 2014 [cit. 2014-04-23]. Dostupné z: http://shop.chessbase.com/en/products/mega_database_2014

[15] MICROSFT. *Microsoft Developer Network: Data Types (Transact-SQL)* [online]. 2014 [cit. 2014-04-12]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms187752.aspx

[16] ORACLE. *JavaServer Pages Standard Tag Library 1.1 Tag Reference* [online]. 2003 [cit. 2014-04-16]. Dostupné z: http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/

Seznam zkratek

API	Application Programming Interface, rozhraní pro programování aplikací
ASC	Ascending, řazení vzestupně
ASCII	American Standard Code for Information Interchange, americký standardní kód pro výměnu informací
CSS	Cascade Style Sheet, kaskádové styly
DBMS	DataBase Management System, systém pro řízení báze dat
DCL	Data Control Language, příkazy pro řízení přístupu
DDL	Data Definition Language, jazyk pro definici dat
DESC	Descending, řazení sestupně
DGT	Digital, digitální (část názvu společnosti)
DML	Data Manipulation Language, jazyk pro manipulaci s daty
ECO	Encyclopaedia of Chess Openings, encyklopedie šachových zahájení
EE	Enterprise Edition, edice podnikových aplikací
EJB	Enterprise Java Beans, podnikové Java Beany
FK	Foreign Key, cizí klíč
HTML	HyperText Markup Language, hypertextový značkovací jazyk
HTTP	HyperText Transfer Protocol, internetový protokol
IBM	International Business Machine, přední světová společnost v IT
IDE	Integrated Development Environment, integrované vývojové prostředí
J2SE	Java 2 Standard Edition, označení platformy programovacího jazyka Java
JAR	Java ARchive, kompresní souborový formát
JCP	Java Community Process, proces k vývoji různých částí Javy

JDBC	Java DataBase Connectivity, rozhraní pro přístup k relačním databázím v Javě
JDK	Java Development Kit, základní nástroje pro vývoj aplikací v Javě
JNDI	Java Naming and Directory Interface, rozhraní pro jmenné a adresářové služby v jazyku Java
JPA	Java Persistance API, framework Javy umožňující objektově relační mapování
JSF	JavaServer Faces, komponentově založená technologie Java EE
JSP	JavaServer Pages, technologie pro vytváření dynamických webových stránek v Javě
JSTL	Java Standard Tag Library, knihovna standardních značek v Javě
JVM	Java Virtual Machine, virtuální stroj Javy
MB	MegaByte, jednotka kapacity informace
NT	New Technology, nová technologie, označení řady Microsoft Windows
OOP	Object-Oriented Programming, objektově orientované programování
PC	Personal Computer, osobní počítač
PGN	Portable Game Notation, formát souboru pro zápis šachové partie
PK	Primary Key, primární klíč
PNG	Portable Network Graphic, formát souboru pro internetovou grafiku
SQL	Structured Query Language, strukturovaný dotazovací jazyk
SŘBD	Systém Řízení Báze Dat
STR	Seven Tag Rooster, seznam sedmi tagů, pravidlo ve specifikaci formátu PGN
TCC	Transaction Control Commands, příkazy pro řízení transakcí
URL	Uniform Resource Locator, jednotný lokátor zdrojů
WWW	World Wide Web, celosvětová síť počítačů
XML	eXtensible Markup Language, rozšiřitelný značkovací jazyk

Prohlášení o využití výsledků diplomové práce

Prohlašuji, že

- jsem byl seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;

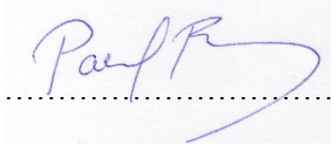
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3);

- souhlasím s tím, že diplomová práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že bibliografické údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO;

- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;

- bylo sjednáno, že užít své dílo, diplomovou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 25. 4. 2014

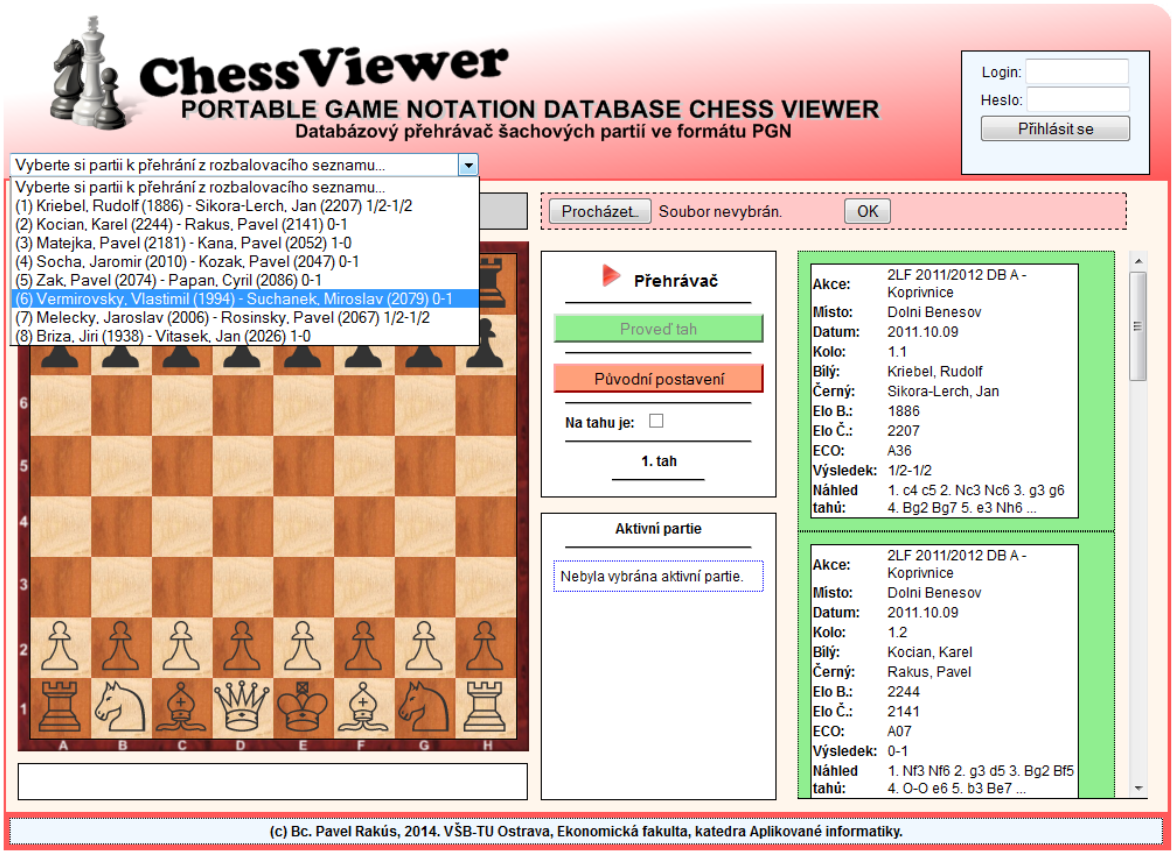


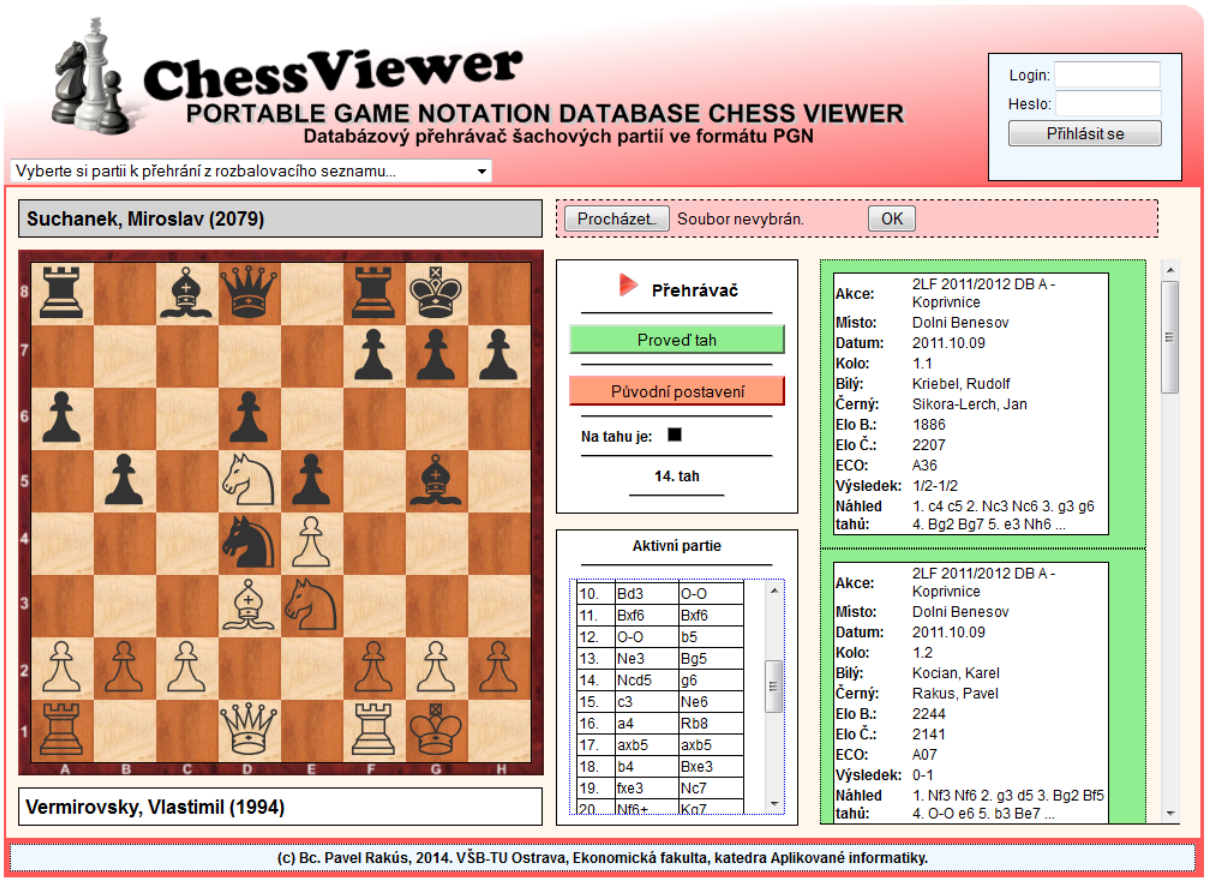
……

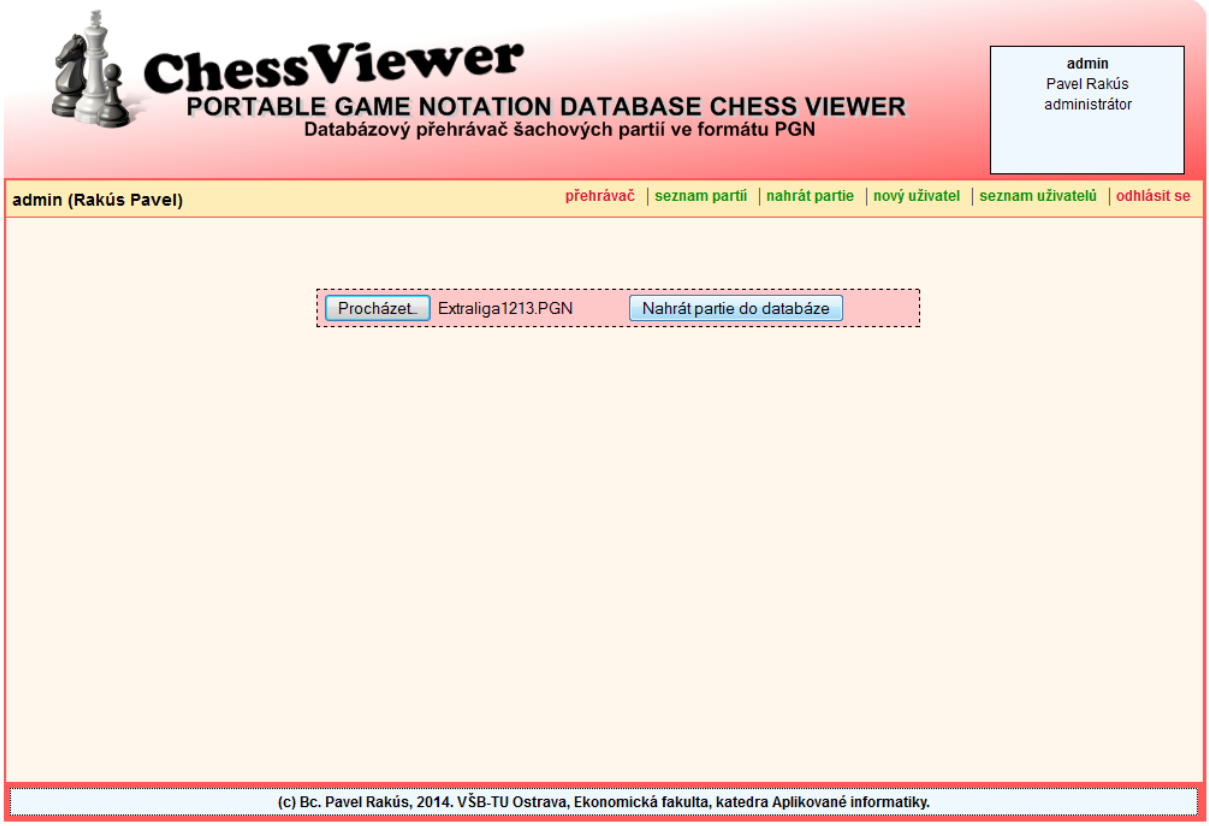
Bc. Pavel Rakús

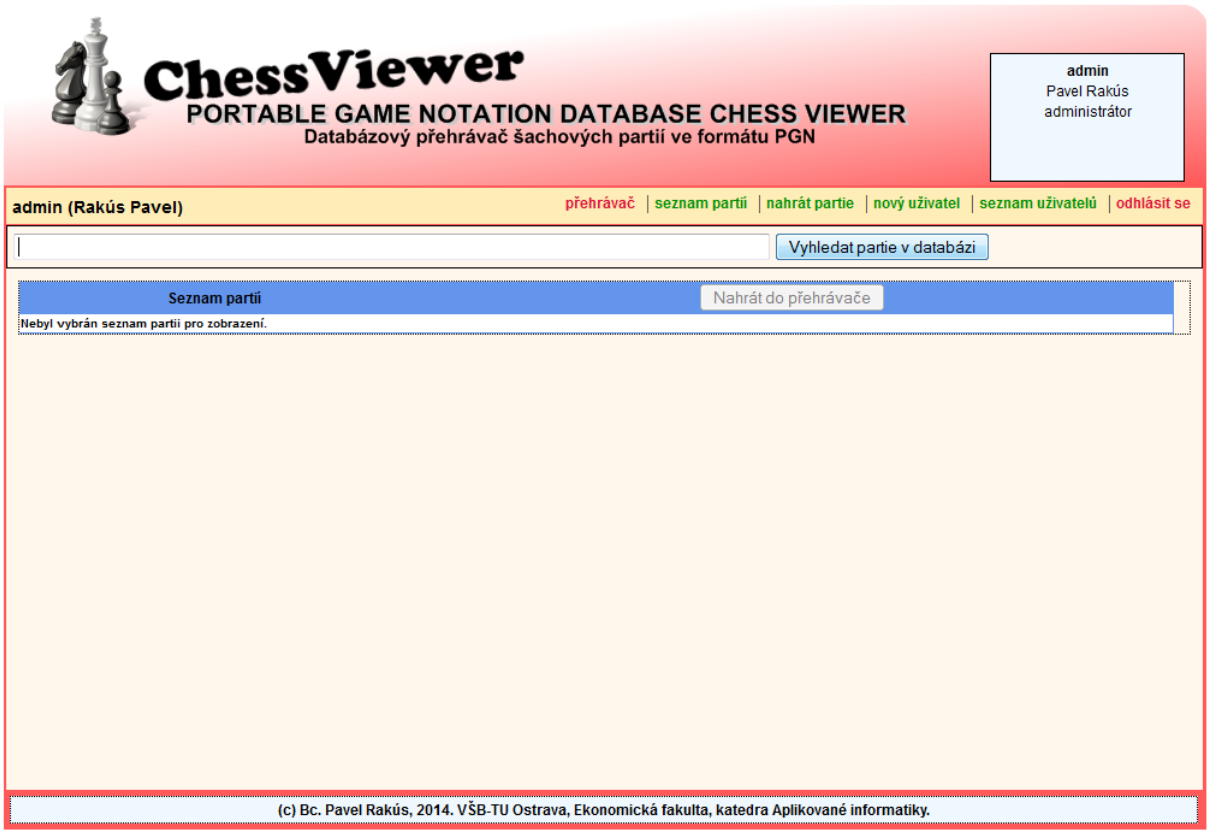
Seznam příloh

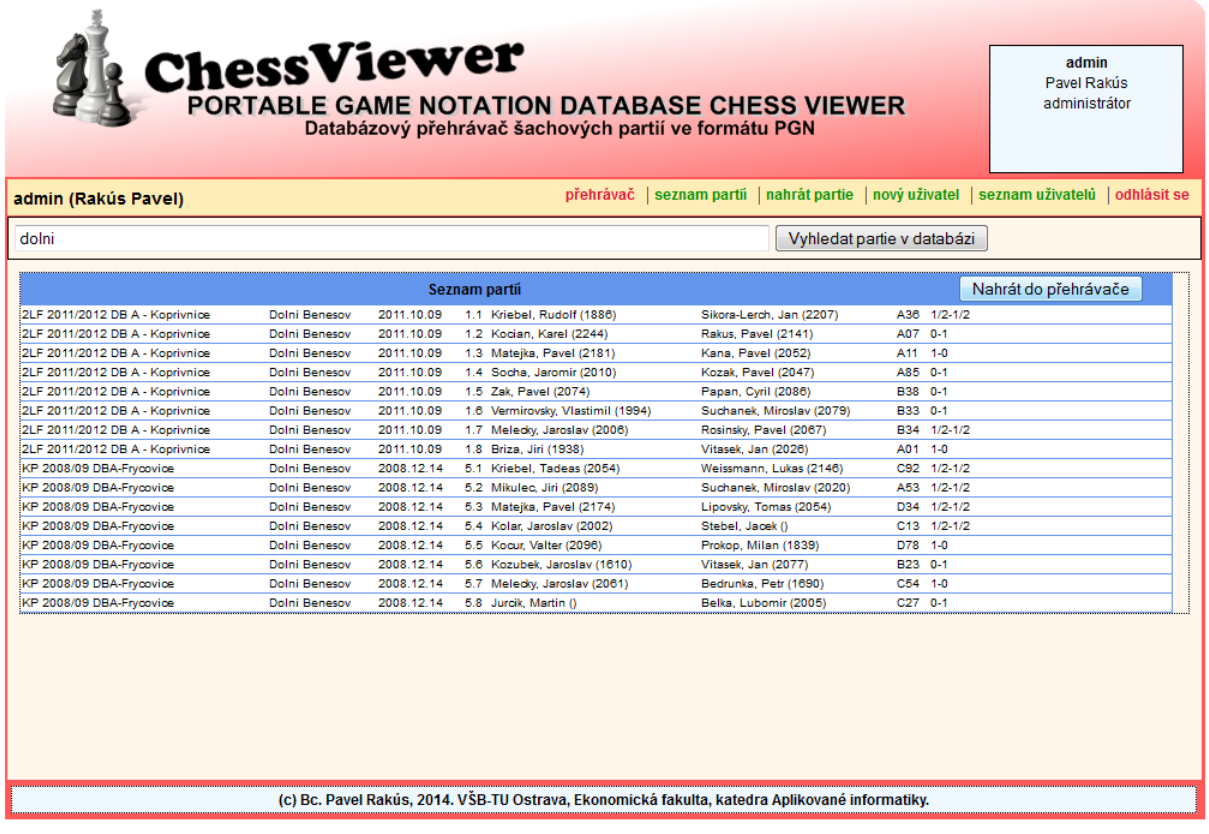
Příloha č. 1 - Obrazová příloha




ChessViewer
PORTABLE GAME NOTATION DATABASE CHESS VIEWER
Databázový přehrávač šachových partií ve formátu PGN
Login:
Heslo:
Přihlásit se
Vyberte si partii k přehrání z rozbalovacího seznamu...
Suchanek, Miroslav (2079)
Procházet..
Soubor nevybrán.
OK
Přehrávač
Proveď tah
Původní postavení
Na tahu je:
14. tah
Aktivní partie
10. Bd3 O-O
11. Bxf6 Bxf6
12. O-O b5
13. Ne3 Bg5
14. Ncd5 g6
15. c3 Ne6
16. a4 Rb8
17. axb5 axb5
18. b4 Bxe3
19. fxe3 Nc7
Akce: 2LF 2011/2012 DB A - Koprivnice
Místo: Dolni Benesov
Datum: 2011.10.09
Kolo: 1.1
Bílý: Kriebel, Rudolf
Černý: Sikora-Lerch, Jan
Elo B.: 1886
Elo Č.: 2207
ECO: A36
Výsledek: 1/2-1/2
Náhled tahů: 1. c4 c5 2. Nc3 Nc6 3. g3 g6 4. Bg2 Bg7 5. e3 Nh6 ...
Akce: 2LF 2011/2012 DB A - Koprivnice
Místo: Dolni Benesov
Datum: 2011.10.09
Kolo: 1.2
Bílý: Kocian, Karel
Černý: Rakus, Pavel
Elo B.: 2244
Elo Č.: 2141
ECO: A07
Výsledek: 0-1
Náhled tahů: 1. Nf3 Nf6 2. g3 d5 3. Bg2 Bf5 4. O-O e6 5. b3 Be7 ...
Vermirovsky, Vlastimil (1994)
(c) Bc. Pavel Rakús, 2014. VŠB-TU Ostrava, Ekonomická fakulta, katedra Aplikované informatiky.


ChessViewer
PORTABLE GAME NOTATION DATABASE CHESS VIEWER
Databázový přehrávač šachových partií ve formátu PGN
admin
Pavel Rakús
administrátor
admin (Rakús Pavel)
přehrávač | seznam partií | nahrát partie | nový uživatel | seznam uživatelů | odhlásit se
Procházet..
Extraliga1213.PGN
Nahrát partie do databáze
(c) Bc. Pavel Rakús, 2014. VŠB-TU Ostrava, Ekonomická fakulta, katedra Aplikované informatiky.


ChessViewer
PORTABLE GAME NOTATION DATABASE CHESS VIEWER
Databázový přehrávač šachových partií ve formátu PGN
admin
Pavel Rakús
administrátor
admin (Rakús Pavel)
přehrávač | seznam partií | nahrát partie | nový uživatel | seznam uživatelů | odhlásit se
Vyhledat partie v databázi
Seznam partií
Nahrát do přehrávače
Nebyl vybrán seznam partii pro zobrazení.
(c) Bc. Pavel Rakús, 2014. VŠB-TU Ostrava, Ekonomická fakulta, katedra Aplikované informatiky.


ChessViewer
PORTABLE GAME NOTATION DATABASE CHESS VIEWER
Databázový přehrávač šachových partií ve formátu PGN
admin
Pavel Rakús
administrátor
admin (Rakús Pavel)
přehrávač | seznam partií | nahrát partie | nový uživatel | seznam uživatelů | odhlásit se
dolni
Vyhledat partie v databázi
Seznam partií
Nahrát do přehrávače
2LF 2011/2012 DB A - Koprivnice Dolni Benesov 2011.10.09 1.1 Kriebel, Rudolf (1886) Sikora-Lerch, Jan (2207) A36 1/2-1/2
2LF 2011/2012 DB A - Koprivnice Dolni Benesov 2011.10.09 1.2 Kocian, Karel (2244) Rakus, Pavel (2141) A07 0-1
2LF 2011/2012 DB A - Koprivnice Dolni Benesov 2011.10.09 1.3 Matejka, Pavel (2181) Kana, Pavel (2052) A11 1-0
2LF 2011/2012 DB A - Koprivnice Dolni Benesov 2011.10.09 1.4 Socha, Jaromir (2010) Kozak, Pavel (2047) A85 0-1
2LF 2011/2012 DB A - Koprivnice Dolni Benesov 2011.10.09 1.5 Zak, Pavel (2074) Papan, Cyril (2086) B38 0-1
2LF 2011/2012 DB A - Koprivnice Dolni Benesov 2011.10.09 1.6 Vermirovsky, Vlastimil (1994) Suchanek, Miroslav (2079) B33 0-1
2LF 2011/2012 DB A - Koprivnice Dolni Benesov 2011.10.09 1.7 Melecky, Jaroslav (2006) Rosinsky, Pavel (2067) B34 1/2-1/2
2LF 2011/2012 DB A - Koprivnice Dolni Benesov 2011.10.09 1.8 Briza, Jiri (1938) Vitasek, Jan (2026) A01 1-0
KP 2008/09 DBA-Frycovice Dolni Benesov 2008.12.14 5.1 Kriebel, Tadeas (2054) Weissmann, Lukas (2146) C92 1/2-1/2
KP 2008/09 DBA-Frycovice Dolni Benesov 2008.12.14 5.2 Mikulec, Jiri (2089) Suchanek, Miroslav (2020) A53 1/2-1/2
KP 2008/09 DBA-Frycovice Dolni Benesov 2008.12.14 5.3 Matejka, Pavel (2174) Lipovsky, Tomas (2054) D34 1/2-1/2
KP 2008/09 DBA-Frycovice Dolni Benesov 2008.12.14 5.4 Kolar, Jaroslav (2002) Stebel, Jacek () C13 1/2-1/2
KP 2008/09 DBA-Frycovice Dolni Benesov 2008.12.14 5.5 Kocur, Valter (2096) Prokop, Milan (1839) D78 1-0
KP 2008/09 DBA-Frycovice Dolni Benesov 2008.12.14 5.6 Kozubek, Jaroslav (1610) Vitasek, Jan (2077) B23 0-1
KP 2008/09 DBA-Frycovice Dolni Benesov 2008.12.14 5.7 Melecky, Jaroslav (2061) Bedrunka, Petr (1690) C54 1-0
KP 2008/09 DBA-Frycovice Dolni Benesov 2008.12.14 5.8 Jurcik, Martin () Belka, Lubomir (2005) C27 0-1
(c) Bc. Pavel Rakús, 2014. VŠB-TU Ostrava, Ekonomická fakulta, katedra Aplikované informatiky.